COMPUTER SIMULATIONS OF REALISTIC THREE-DIMENSIONAL MICROSTRUCTURES

A Dissertation
Presented to
The Academic Faculty

by

Yuxiong Mao

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Materials Science and Engineering

Georgia Institute of Technology

May 2010

COMPUTER SIMULATIONS OF REALISTIC THREE-DIMENSIONAL MICROSTRUCTURES

Approved by:

Dr. Arun M. Gokhale, Advisor
School of Materials Science & Engineering
Georgia Institute of Technology

Dr. David Frost
School of Civil & Environmental Engineering
Georgia Institute of Technology

Dr. Meilin Liu
School of Materials Science & Engineering
Georgia Institute of Technology

Dr. Burton R Patterson
Department of Materials Science & Engineering
University of Alabama at Birmingham

Dr. Min Zhou
School of Mechanical Engineering
Georgia Institute of Technology

Date Approved: February 11, 2010

ACKNOWLEDGEMENTS

I would like to thank all people who have helped and inspired me during my time at Georgia Tech.

Foremost, I would like to express my sincere gratitude to my advisor, Professor Arun Gokhale, for his continuous support of my study and research, for his patience, motivation, enthusiasm, and immense knowledge. Completion of this dissertation would not have been possible without his invaluable guidance.

I am indebted to my colleagues in Dr. Gokhale's research group, especially, Gautam Patel, Harpreet Singh, Arun Sreeranganathan, Joel Harris, Soon Gi Lee, Scott Lieberman, Asim Tewari, and Shenjia Zhang. Their contributions and collaborative efforts over the last few years are greatly appreciated. I must also thank my committee members, Dr. David Frost, Dr. Meilin Liu, Dr. Burton R Patterson, and Dr. Min Zhou for their guidance and support of this work.

Financial support for this research came through research grants from Division of Materials Research, U.S. National Science Foundation (NSF grants DMR 0404668 and 0813636), and Air Force Office of Scientific Research (AFOSR grants FA95550-05-1-0062 and F49620-01-1-0045). The support is gratefully acknowledged.

Last but most importantly, I would like to thank my family, especially my wife Xinzheng, for her unconditional love, encouragement, understanding and endless patience, and my parents back home, for their consistent support throughout my life.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

SUMMARY

A novel and efficient methodology is developed for computer simulations of *realistic* two-dimensional (2D) and three-dimensional (3D) microstructures. The simulations incorporate realistic 2D and 3D complex morphologies/shapes, spatial patterns, anisotropy, volume fractions, and size distributions of the microstructural features statistically similar to those in the corresponding real microstructures. The methodology permits simulations of sufficiently large 2D as well as 3D microstructural windows that incorporate short-range (on the order of particle/feature size) as well as long-range (hundred times the particle/feature size) microstructural heterogeneities and spatial patterns at high resolution. The utility of the technique has been successfully demonstrated through its application to the 2D microstructures of the constituent particles in wrought Al-alloys, the 3D microstructure of discontinuously reinforced Al-alloy (DRA) composites containing SiC particles that have complex 3D shapes/morphologies and spatial clustering, and 3D microstructure of boron modified Ti-6Al-4V composites containing fine TiB whiskers and coarse primary TiB particles. The simulation parameters are correlated with the materials processing parameters (such as composition, particle size ratio, extrusion ratio, extrusion temperature, etc.), which enables the simulations of rational virtual 3D microstructures for the parametric studies on microstructure-properties relationships. The simulated microstructures have been implemented in the 3D finite-elements (FE)-based framework for simulations of micro-mechanical response and stress-strain curves. Finally, a new unbiased and assumption free dual-scale virtual cycloids probe for estimating surface area of 3D objects constructed by 2D serial section images is also presented.

CHAPTER 1

INTRODUCTION

1.1 Problem Formulation and Research Objectives

Material microstructures usually contain features at numerous different length scales. The microstructural features may consist of regions of phases of different compositions and/or crystal structures, grains of different orientations, voids, inclusions, domains of different electrical or magnetic polarizations, or structural defects. Modeling and simulations of microstructures at length scales of interest is an important aspect of computational materials science. Microstructure models are utilized for mathematical/statistical representation of microstructural geometry [1-11]. Simulated microstructures are often used as microstructural windows in the computational models and simulations of materials behavior [12-20]. For reliable predictions of material properties and performance using such techniques, it is necessary that a simulated (or modeled) microstructure correctly represent the relevant microstructural geometry, because the microstructural geometry significantly influences material properties. Some important geometric characteristics of real material microstructures are as follows:

- Microstructures are usually three-dimensional (3D)
- Microstructures are stochastic
- Microstructural features (particles, voids, etc) typically have complex (often non-convex) shapes and morphologies that cannot be satisfactorily represented by simple shapes such as spheres and ellipsoids

- Spatial arrangements of microstructural features are often not uniform-random as spatial clustering of features is commonly observed
- Microstructural features often have partially anisotropic orientations.

It is important to recognize that the particle/feature shapes and morphologies, spatial arrangements and clustering of features, and morphological anisotropy also significantly affect numerous material properties. For example, local stresses and strains around the particles/features in a microstructure significantly depend on the particle/feature shapes, which can in turn affect the mechanical properties of the material [21]. Therefore, in addition to the fundamental microstructural parameters such as relative amounts of different phases (volume fractions) and their number densities, etc., the microstructure simulations should incorporate realistic 3D nature of microstructure, realistic particle shapes/morphologies, realistic spatial arrangements, and realistic partial anisotropy of the microstructural features of interest.

The microstructure simulations techniques can be classified into three groups, namely, the methods based on ab-initio and molecular dynamics (MD) simulations [22], the techniques based on microstructure evolution that incorporate thermodynamics of phases and kinetics of phase transformations [23-24], and the methodologies based on the geometric simulations of microstructures [1-10]. The present research concerns the methodologies based on geometric microstructure simulations. Such techniques are particularly attractive for simulations of heterogeneous microstructures of composite materials, inclusions in ferrous and nonferrous alloys, and voids in cast microstructures.

Numerous geometric simulations of model random heterogeneous material microstructures have been reported in the literature [1-20]. Important algorithms for

geometric computer simulations of microstructures include random sequential adsorption (RSA) algorithm [1, 25], Metropolis algorithm [26], Boolean schemes [27], Gaussian random fields [28], simulated annealing process [29], Gibbs process [29], Voronoi tessellations [3, 30-31], and Monte-Carlo based techniques [32]. However, these methodologies (i) assume idealized simple particle/features shapes such as spheres and ellipsoids in 3D and circles and ellipses in 2D or incorporate unrealistic arbitrary digitized particle shapes, and/or (ii) assume that all features are of the same shape and size, and/or (iii) assume uniform-random spatial arrangement of the features, and/or (iv) assume isotropic (or completely anisotropic) morphological orientations of the microstructural features. In majority of cases, the detailed geometric attributes (for example correlation functions, radial distribution functions, orientation distribution functions, etc.) of the simulated microstructures are not compared (and matched) with the *experimentally measured* attributes of the corresponding *real* microstructures to ensure that the simulated microstructure is representative of the corresponding real microstructure. Further, in many cases, the simulated microstructural segments are too small (for example, simulations containing only ten microstructural features of interest) to account for important long-range spatial patterns and heterogeneities in the microstructure, or to serve as representative volume elements (RVEs) of the microstructure in computational models for material behavior. Such microstructure simulations are not likely to be useful for *quantitative* predictions of the mechanical and physical properties of complex real material microstructures. Therefore, there is a need to develop techniques for simulations of realistic complex 3D microstructures that (1) incorporate realistic complex particle/feature shapes, (2) allow controlled non-

uniformities/clustering in spatial distributions of features, (3) permit partial anisotropic morphological orientations of microstructural features, (4) closely match *experimentally measured* attributes (spatial correlation functions, orientation distributions, size and shape distributions, volume fraction, etc.) of the corresponding real microstructures, and (5) efficiently generate sufficiently large segments of microstructure that contain short-range (on the order of particle/feature size), intermediate-range (five to ten times particle/feature size), and long-range (few hundred times the particle/feature size) microstructural heterogeneities and spatial patterns. The present research utilizes a combination of digital image processing techniques and computer simulations to develop an efficient and general methodology for representation and simulations of such realistic three-dimensional microstructures. The present research draws from and extends similar methodology recently developed at Georgia Tech for simulations of 2D microstructures [33]. The specific research objectives are as follows.

- Development of digital image processing based techniques for incorporation of realistic complex 3D particle/feature morphologies in the 3D microstructure simulations.
- Development of stereology and image analysis based procedures for unbiased and efficient estimation of statistical correlation functions as well as local metric properties needed for 3D microstructure representation.
- Development and applications of Monte Carlo based simulation techniques to simulate large segments of 3D microstructures whose two-point microstructural correlation functions and lineal path probability distribution functions match those of the corresponding real microstructures.

- Demonstration of quantitative correlations of microstructure simulation parameters with the material processing conditions and utilization of such correlations for simulations of rational virtual microstructures of the materials that have not been fabricated.

The methodology has been developed through its applications to microstructures of the following materials.

i. Discontinuously reinforced aluminum alloy (DRA) composites having different extents of SiC particle reinforcement volume fractions, spatial clustering, and anisotropy,
ii. Boron modified titanium alloys and composites,
iii. Coarse constituent particles in wrought aluminum alloys.

The above materials have been used in the present work because (i) in these microstructures the long-range microstructural patterns exist up to length scales of 500 μm or so, (ii) these microstructures exhibit spatial clustering and anisotropy, (iii) the microstructural features have complex shapes/morphologies, and (iv) extensive quantitative microstructural data on these microstructures are available in the research group.

1.2 Organization of Thesis

This thesis is organized into six chapters. Each chapter, for the most part, is self-contained and focuses on a specific aspect of the research. Chapter 2 details the background on the current microstructure characterization and representation techniques, as well as the state of the art in the field of the microstructure modeling and simulations. Chapter 3 presents

a methodology for computer simulation of realistic two-dimensional microstructures having realistic particle/feature shapes/morphologies, spatial clustering, and anisotropy, which has been applied to the 2D microstructures of coarse constituent particles in wrought aluminum alloys. Chapter 4 is the central part of the present research. In this chapter, the 2D simulation technique is further developed and extended to simulations of realistic two-phase 3D microstructures. This 3D realistic microstructural simulation methodology is presented through its application to the 3D microstructures of discontinuously reinforced aluminum alloy composites and boron modified titanium alloy composites. Chapter 5 introduces a novel unbiased dual-scale virtual cycloid for surface area estimations of 3D digital objects. Finally, Chapter 6 concludes this thesis with a summary of the completed research and recommendations for future work.

CHAPTER 2

BACKGROUND AND LITERATURE REVIEW

The central objective of the present research is to develop an efficient and general stereology and digital image processing based methodology for representation and simulations of realistic 3D microstructures. The research draws from the current procedures for quantitative characterization and mathematical/statistical representation of microstructures and digital image analysis techniques. Therefore, background on the current microstructure characterization and representation techniques is presented in the next section. The research involves geometric simulations of microstructures, and therefore, the subsequent section gives background and critical analysis of the current microstructure simulation methods.

2.1 Quantitative Characterization and Representation of Microstructural Geometry

Microstructures are of stochastic nature. Therefore, microstructures can be quantitatively characterized and represented only by using mean values and distributions of the relevant statistical descriptors. Each microstructural feature (particle, pore, grain, etc) has associated with it, the "local" metric properties such as volume, surface area, curvature, etc. The global metric properties corresponding to these local metric properties are volume fraction (V_V), total surface area per unit volume (S_V), total length per unit volume (L_V), and the integral mean curvature (M_V), etc. These first order (or fundamental) metric global properties of 3D microstructure (which are also classical Minkowski functionals [34] of integral geometry) can be estimated in an unbiased and efficient manner from the measurements performed on 2D metallographic sections using well

established stereological techniques, which are described in detail elsewhere [35-36]. Modern efficient design-based stereological methods can be used for unbiased and efficient estimation of these microstructural parameters in partially anisotropic microstructures [37]. These first order global microstructural parameters constitute important input for simulation of realistic microstructures.

In addition to the local metric properties, each microstructural feature has also associated with it, a Euler-Poincaré characteristic [38], which gives rise to mean topological global properties such as number densities of features (N_V) and their topological connectivity (as represented by the first Betti number) in the 3D microstructure. Unfortunately, these 3D topological properties cannot be estimated from measurements performed on independent 2D metallographic sections, unless the particles/grains have a known simple convex shape [39]. Consequently, in the past, reliable estimation of number density was problematic. In 1984, Sterio [40] showed that the number density of features in a 3D microstructure could be estimated using a test probe (called disector) consisting of two parallel metallographic planes that are a small distance apart (i.e., two closely spaced serial sections). Recently [41], a combination of this sampling principle and modern image analysis techniques has led to an efficient stereological method for unbiased estimation of number density of microstructural features in 3D from measurements performed on digital images of two closely spaced large-area high-resolution montage serial sections, called large area disector (LAD). The microstructural parameters such as N_V are required for computer simulations of realistic microstructures. Unfortunately, efficient techniques have not been yet developed for

unbiased estimation of 3D topological connectivity. Reconstruction of large segments of 3D microstructure is required for estimation of such attributes.

Finally, each microstructural feature has also associated with it, a morphological orientation and a location in the 3D microstructural space. Distribution of relative morphological orientations of the microstructural features gives rise to morphological orientation distribution function (MODF) that describes orientation distribution of local surface normals at the particle/feature boundaries and mathematically represents the morphological anisotropy of the microstructure. The MODF of 3D microstructural surfaces can be estimated from the stereological measurements that can be performed on 2D metallographic sections [42].

The distribution of relative locations of microstructural features is manifested in the spatial patterns, correlations, clustering, spatial affinity, short and long range interactions, pair correlations and higher order correlations, microstructural gradients, segregations, etc. Theoretical statistics literature contains a large number of contributions that deal with the statistics of spatial point patterns and the quantitative descriptors that reflect various attributes of the spatial arrangement of ensembles of features in one-, two-, and 3D space [36, 43-45]. K-function, radial distribution function (or so called G-function), two-point correlation function and higher order correlation functions, first, second, and higher order nearest neighbor distribution functions, surface to surface distance distributions [46], Voronoi [31], Dirichlet [47], Delaunay [48], and other forms of tessellations[49], Q-mode factor [50], clustering parameters [4, 51], and affinity parameters [52] are some of the statistical descriptors of spatial arrangement of features in microstructures.

The theoretical models, numerical simulations, and experimental observations indicate that spatial microstructural distance distributions affect numerous properties such as connectivity [53], percolation thresholds for rigidity of sintered compacts [54], electrical conductivity of polymer matrix composites [2], and particle agglomerations [55], etc. The finite elements (FE)-based simulations of mechanical response of computer simulated microstructures [48, 56] and 2D digital microstructural images [56], as well as of 3D digital microstructural images reconstructed from serial sections [57] show that the spatial arrangements of microstructural features affect micro-mechanical response of multi-phase microstructures. The experimental observations and theoretical computations also demonstrate that the spatial arrangements of microstructural features affect damage evolution [14, 58-59], crack path and crack growth [60], flow stress [61], ductility [62], fracture resistance [63], superplastic behavior [64], and numerous other mechanical properties. Therefore, it is essential to incorporate realistic spatial arrangements of microstructural features in the computer simulated microstructures. Important statistical descriptors of spatial arrangement of microstructural features are described as follows.

2.1.1 Nearest Neighbor Distance Distributions

The nearest neighbor distribution function captures the information about the spatial arrangement of features in short range [17, 36]. The 3D nearest neighbor distribution function is the probability density function $\psi(r)$, such that $\psi(r)dr$ is equal to the probability that there is no feature center in a sphere of radius r around a typical microstructural feature of interest, and there is at least one feature center in the spherical shell of radii r and $r + dr$. The average nearest neighbor distance $\bar{r}$ is given as follows:

$$\bar{r} = \int_0^{\infty} r \cdot \psi(r) dr \qquad \text{Eq. 1}$$

For a uniform–random distribution of zero dimensional points of intensity N_V in the 3D space, the Poisson process statistics yields the following expressions for the distribution function $\psi(r)$ [45]:

$$\psi(r) = 4\pi r^2 N_V \exp(-\frac{4}{3} r^3 N_V) \qquad \text{Eq. 2}$$

It is shown that in uniform random microstructure containing impenetrable poly-dispersed spherical particles that have no spatial size correlations, the mean nearest neighbor distance depends only on the number density, sphere volume fraction, and coefficient of variation of the sphere size distribution; it is not sensitive to the other attributes of the size distribution function such as skewness [17].

2.1.2 K-Function and Radial Distribution Function

The K-function and the radial distribution function (also called G-function) characterize short-, intermediate-, and long-range spatial arrangement of particle centers [65]. The average number of other particle centers in a test sphere of radius r drawn around a typical particle is called K-function, $K_V(r)$, for the 3D microstructure. The radial distribution function $G_V(r)$ is related to the derivative of the K-function, and it is defined as follows [37]:

$$G_V(r) = \frac{1}{4\pi r^2 N_V} \cdot \frac{dK_V(r)}{dr} \qquad \text{Eq. 3}$$

The radial distribution function can be thought as the degree of deviation from randomness. For a set of completely random distribution of points, $G_V(r)$ is equal to 1. A value higher than 1 indicates clustering of points and a value less than 1 indicates

repulsion among the points. The radial distribution function has been utilized for realistic computer simulation of microstructure of continuous aligned fiber reinforced ceramic matrix composite [4], where the transverse 2D metallographic section essentially contains all the information concerning the spatial arrangement of fibers. Although, K-function and radial distribution functions are important statistical descriptors of spatial arrangement of features, these distribution functions for 3D microstructure cannot be estimated from any measurements performed on independent 2D metallographic sections. On the other hand, the two- and three-point correlation functions of 3D microstructure can be estimated from the measurements performed on 2D metallographic sections. These statistical distribution functions are described in the next sub-section.

2.1.3 N-point Correlation Functions

Consider placement of a polyhedron having n-vertices at random locations in the 3D microstructure of interest containing two phases, namely, phase-1 (which may be particles, voids, inclusions, etc) and phase-2 (matrix). The probability that all n vertices of the polyhedron are contained in phase-1 is an n-point correlation function that varies with the size, shape, and orientation of the polyhedron in the 3D space of the microstructure [44]. One can similarly define another *n*-point correlation function that represents the probability that all the n-vertices of the polyhedron are in phase-2, and so on. Thus, one can formulate one-, two-, three-, …, and *n*-point correlation functions representing the corresponding probabilities. These statistical correlation functions implicitly contain information concerning the first order microstructural parameters such as volume fractions, number densities, and size distributions, as well as spatial arrangements and morphological anisotropy of the features in 3D microstructure, and

therefore, they are useful for mathematical representation of microstructures. Statistical mechanics theories link the correlation functions of a heterogeneous material microstructure to the properties such as elastic constants and thermal conductivity [15-16, 66].

The lowest order correlation function is the 1-point correlation function, which is the probability that a randomly placed point in a 3D microstructure is contained in a given phase, and that is precisely equal to the volume fraction of that phase. For a two-phase microstructure, a two-point correlation function $P_{ij}(r, \theta, \phi)$ is the probability that a straight line of length r and angular orientation (θ, ϕ) randomly placed in a 3D microstructure is such that its first end is in the phase i (where, i = 1 or 2) and the second end is contained in the phase j (where, j = 1 or 2), as illustrated in Figure 2.1.

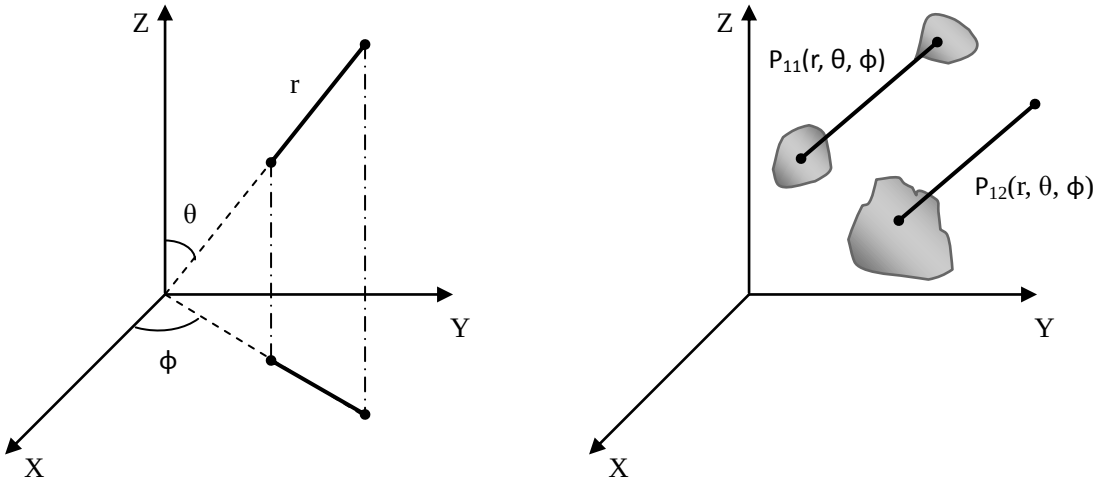


Figure 2.1: Schematic of two-point correlation function

Note that the probability associated with a two-point correlation function only concerns the events at the end points of the line. For a two-phase microstructure, there are four possible two-point correlation functions, namely, $P_{11}(r, \theta, \phi)$, $P_{22}(r, \theta, \phi)$, $P_{12}(r, \theta, \phi)$, and $P_{21}(r, \theta, \phi)$. However, only one of the four two-point correlation functions is independent [66-67]. As r approaches zero, $P_{ii}(r, \theta, \phi)$ approaches the value equal to the volume fraction of the i phase. On the other hand, as r goes to infinity, $P_{ii}(r, \theta, \phi)$ approaches the value equal to the square of the volume fraction of the i phase. At other values of the length of the line r, $P_{ii}(r, \theta, \phi)$ depends on the first order global properties of microstructure as well as on the spatial arrangement of the features and their anisotropy. Therefore, the two-point correlation function contains information on the first order properties such as volume fraction and surface area per unit volume, size and shape distributions of the features, morphological anisotropy, as well spatial arrangements of microstructural features. Consequently, in the present research, two-point correlation functions are extensively used for microstructure representation. The two-point correlation functions of the simulated microstructures are matched closely with the two-point function data on the corresponding real microstructures to ensure that the simulated microstructures are statistically similar to the corresponding real microstructures.

Recently, a robust digital image analysis and stereology based technique has been developed for estimation of direction dependent two point correlation functions of any 3D microstructure from the measurements performed on vertical metallographic sections [68]. The technique permits precise and automatic estimation of the two-point correlation functions at distances ranging from 1 μm to 1000 μm (or more if needed) at a resolution

on the order of 0.5 μm; the correlation functions can be estimated for all discrete line orientations in the vertical planes.

A three-point correlation function $P_{111}(r_1, r_2, r_3, \theta, \phi, \alpha)$ is the probability that a triangle of having sides of length r_1, r_2, r_3 and orientation α randomly placed in a metallographic plane of orientation (θ, ϕ) is such that its all three vertices are contained in the particulate phase (phase-1). A three-point correlation function contains more detailed microstructural information than corresponding two-point correlation function. However, there is no well-developed experimental technique for unbiased and efficient estimation of three-point correlation functions. On the other hand, lineal path probability distributions that represent the probability that a test line segment of given length and orientation is completely contained in the phase of interest have microstructural geometric information that is not reflected in these *n*-point correlation functions, and they can be estimated efficiently from 2D vertical sections [69]. The details of lineal path probability distributions are described in the following sub-section.

2.1.4 Lineal Path Probability Distributions

Consider a microstructure containing two phases, namely phase-1 (particles) and phase-2 (matrix). The lineal path distribution function $L_{11}(r, \theta, \varphi)$ is the probability that a randomly located straight line of length r and angular orientation (θ, φ) is completely contained in the phase-1 [70-71]. Similarly, lineal path distribution function $L_{22}(r, \theta, \varphi)$ is the probability that a randomly located straight line of length r and angular orientation (θ, φ) is completely contained in the phase-2. One can also define lineal path distribution function $L_{12}(r, \theta, \varphi)$ as the probability that a randomly located straight line of length r

and angular orientation (θ, φ) intersects the interface between the particles and the matrix at least once [16].

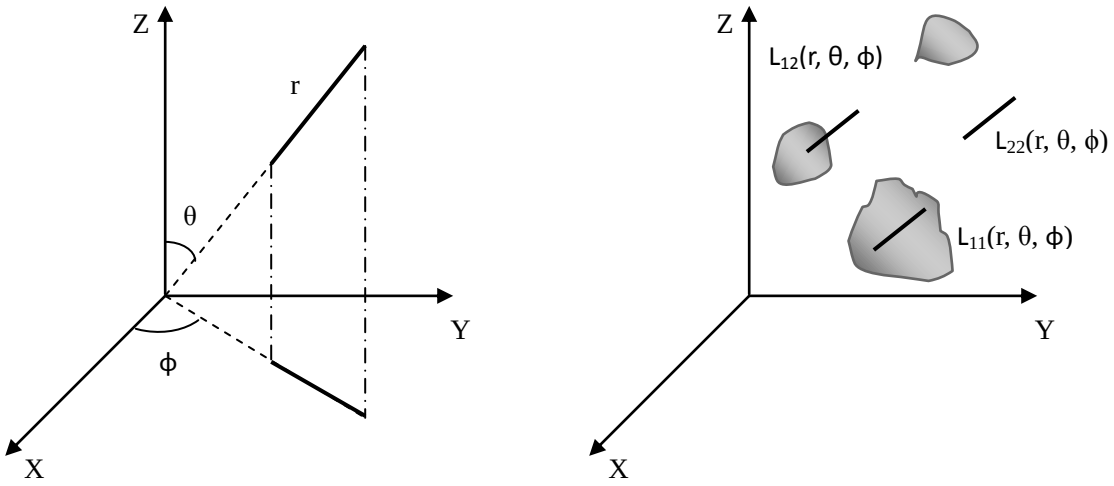


Figure 2.2: Schematic of linear path probability distribution functions

Figure 2.2 shows a schematic for a test line in 3D space and the linear path probability distribution functions. Obviously, for any given values of r, θ and φ, the sum of these three lineal path probability functions must be equal to one. Therefore, only two of the three lineal path probability distributions are independent. Note that L_{11} (r, θ, φ) does not provide any information about the spatial arrangement (clustering, etc) of the particles [70]. On the other hand, the function L_{22} (r, θ, φ) depends on the metric properties of the microstructure, the size, shape, and orientation distribution of the particles, their morphological anisotropy, and the spatial arrangement and clustering of the particles. The lineal path probability functions are independent of the two-point correlation functions, i.e., lineal path probability distributions cannot be calculated from the two-point correlation functions of the same microstructure. This is because the two-

point correlation functions represent the probabilities for the events that only concern the end points of the line, whereas the lineal path probabilities are for the events that concern the complete line segment length. Consequently, lineal path probability distributions provide information about the microstructural geometry that is not contained in the classical *n*-point correlation functions. In the present work, both two-point correlation function and lineal path probability distribution have been utilized for mathematical representation and simulations of microstructures.

2.1.5 Reconstruction of Three-Dimensional Microstructures

Stereological techniques enable estimation of numerous 3D microstructural parameters through observations on 2D sections. Nonetheless, a 2D metallographic section does not contain all of the information concerning the true 3D geometry of the microstructure. For example, information concerning 3D particle shapes/morphologies and topological properties cannot be obtained from independent 2D sections. Therefore, reconstruction and visualization of 3D microstructures are of great interest for understanding such aspects of 3D microstructures. Depending on the material chemistry, processing, and microstructural length scales of interest, an opaque 3D microstructure can be rendered using numerous techniques including serial sectioning [72], focused ion beam (FIB) tomography [73], atom probe tomography [74], magnetic resonance imaging (MRI), and x-ray tomography [75]. The three dimensional microstructural visualization can be achieved either by surface rendering or by volume rendering [72, 76-77]. Surface rendering involves rendering of the iso-surface of the region of interest (ROI) from the volume data, whereas the volume rendering is the rendering of all volume data by specifying opacity and color of each voxel (3D pixel). The surface rendering leads to

reduction in the size of the data set because only the surface data are retained. The surface rendering requires fitting of a surface in the volume data. Numerous algorithms are available for surface rendering, including contour connecting algorithm [78] and marching cube algorithm [79]. In the process of volume rendering, all voxels are visualized by specifying a mapping between rendered image intensity and voxel intensity. Surface rendering is useful for examination of the 3D particle shapes and morphologies, whereas volume rendering is useful for the implementation of the 3D microstructural images in FE-based computations. An efficient and unbiased montage serial sectioning technique has been developed for the reconstruction of large volume (~several mm^3), high resolution (~1 μm) 3D microstructures [33]. The technique involves capturing large-area high-resolution montage images of the metallographic plane after each polishing step, which are subsequently used to reconstruct large 3D volumes. The 3D microstructure reconstructions are useful for extraction of complex 3D particle shapes for implementation in 3D computer simulations of realistic microstructures.

2.2 Current Techniques for Computer Simulations of Microstructures

The length scales of the features and spatial patterns in a microstructure can range from nanometers to millimeters. Numerous modeling techniques are available that can be applied to model microstructure and material behavior at different length and time scales. For example, Molecular dynamics and Quantum mechanical methods are used to model materials at the atomic scale [22], Potts Model methods are applied for meso-scale modeling [80], computational thermodynamics methods can be applied at higher length scales. The scope of each modeling techniques' regime is constantly being expanded as

computational resources are pushed to their limits. Therefore, in the next decade or so, one can expect greater overlap between the phenomena and length/time scales handled by different techniques, making it possible to develop reliable multi-scale modeling techniques. The scope of the present research is on geometric 3D microstructure modeling, where the features and spatial patterns in the range of 1 to 1000 μm are of interest. Therefore, such techniques are presented in more details in subsequent sections.

Numerous geometric simulations of model random heterogeneous material microstructures have been reported in the literature [1-20]. Important algorithms for geometric computer simulations of microstructures include random sequential adsorption (RSA) algorithm [1, 25] , Metropolis algorithm [26], Boolean schemes [27] , Gaussian random fields [28], simulated annealing process [29], Gibbs process [29], Voronoi tessellations [3, 30-31], Dirichlet tessellations [47], cellular automata [81], and Monte-Carlo based techniques [32]. Some of the important simulation algorithms that the present research utilizes are described as follows.

2.2.1 Random Sequential Adsorption (RSA) Algorithm

One of the common methods for simulation uniform random microstructure containing impenetrable particles is random sequential adsorption (RSA) [25], which involves the following rules: (i) objects are placed at uniform-random locations in a 3D volume; (ii) if the last placed object overlaps with any of those already present it is immediately removed; and (iii) otherwise it is permanently fixed. The process usually begins from an empty volume and continues till the desired volume fraction of particles is achieved. There is an upper limit (called jamming limit) on the particulate volume fraction that can be obtained using RSA algorithm. This is because the space available to

place successive particles decreases with the addition of each new particle. In particular, the jamming area fraction for RSA in a 2D plane has been determined as 0.547 [82-83], whereas in the 3D space it has been found to be 0.382 [84] via computer simulations. This process of generation of uniform–random microstructure is very similar to the natural process of adsorption of large particles (proteins, viruses, bacteria, colloids, and macro-molecules) on surfaces formed by phase interfaces. Especially, the two characteristic properties of adsorption, namely, irreversibility of adsorption (in the experimental time scales) and geometric blockage due to previously present particle leading to jamming limit are also present in the RSA algorithm.

A major challenge in the simulation of poly-dispersed size and shape distribution realizations using the RSA algorithm is of avoiding correlations among sizes/shapes. For instance, if one starts the RSA process by putting particles in a descending order of size, it leads to a spatial correlation such that the probability of having a small particle near a large particle is significantly higher than that of having two large particles near to one another. Such correlations can also result in clustering of small particles together. On the other hand, if the particles are placed in the simulation space in an ascending order of size, it gives an inverse size correlation. Therefore, in the present research, wherever RSA algorithm is incorporated, either adequate care is taken to ensure that there are no spatial correlations among particles/features of different sizes and shapes, or the algorithm is appropriately modified to introduce the spatial correlations present in the correspondsvxding real microstructure.

2.2.2 Metropolis Algorithm

Metropolis algorithm (MA) is another method for simulation uniform–random microstructure containing impenetrable objects [26]. MA algorithm can achieve significantly higher particle volume fractions than the corresponding RSA jamming limit. Therefore, it is useful for simulations of uniform random microstructures containing high volume fractions of impenetrable particles.

The MA process is a type of Monte Carlo method that involves the following steps. At the beginning of the simulation, all objects are placed in a 3D lattice such as FCC or BCC and then each object is allowed to take a n-step (n>1000) random walk without causing overlap with other objects. The resulting microstructure has no memory of the initial lattice structure. Since the particles in MA algorithm are allowed to diffuse randomly, they result in ensembles with each possible realization having equal probability. This is unlike the ensembles by RSA algorithm (where the particles are added sequentially and are frozen in space), where open structures are more probable than clustered. This has been demonstrated [85] for a three-particle system, via calculation of probabilities of a typical realization for both MA and RSA. An ensemble of non-interacting hard-particles at thermodynamic equilibrium would have a probability density, which samples the configurational space uniformly; hence, MA is a realization of such a system. Moreover, since the MA algorithm starts with a given lattice arrangement having all the particles, it can be used to simulate much higher particle volume fractions (only limited by the highest packing fraction in that dimension), which is always higher than the highest volume fractions possible by RSA.

2.2.3 Boolean Models

In a Boolean simulation scheme [27], first a set of points (representing particle centers) having a specified mean number density is simulated such that the spatial distribution of the points follows the Poisson point process of specific intensity equal to the mean number density of the points. Next, the objects (particles) of given size and shape distribution are placed randomly at these simulated particle centers. In the Boolean models the particles are permitted to freely overlap to any extent, which is different from the RSA or MA process where the particles are not allowed to overlap. In the realizations of a Boolean model, it is not uncommon to have a particle completely inside another particle. Boolean simulations generate microstructures similar to those encountered in the Johnson-Mehl-Avrami-Kolmogorov model [86] for microstructural evolution.

2.2.4 Cherry-Pit Models

Materials processes often lead to microstructures where the microstructural features may overlap only to a limited extent. The Cherry-Pit model nicely captures such microstructural reality. In the Cherry-Pit model, the features are permitted to overlap only to the specified extent [87]. The extent of permitted overlaps is an important simulation parameter for this model.

2.2.5 Simulations of Spatially Non-Uniform Microstructures

As discussed earlier, the spatial arrangement of microstructural features is often non-uniform and/or non-random. In assessing the effects of inhomogeneities on spatial distribution parameters, a variety of clustered patterns may be considered [8]. One method is to randomly locate points within a number of randomly located clusters of specified size/shape distribution and number density [88]. The degree of clustering can be

specified as the ratio of the number density of particles in the clusters and the global mean number density of the particles. Alternatively, clustered point distributions can be simulated by initially locating "parent" points at random in a given area, and then allocating "offspring" points to each parent [89]. Offspring points are placed about each parent point such that their distances to the parent point are drawn from a normal distribution. In this case, the degree of clustering of the distribution is controlled by the number of parent points and/or the width of the distance distribution around each parent point. Further, diffusion-limited aggregation (DLA) process also can be used to simulate the clustering of particles [90]. In the DLA process, the particles undergo Brownian motion toward the cluster, and stick irreversibly with the cluster when they come into contact with it. The nature of the spatial clustering in the microstructure determines which clustering model is appropriate for a given material.

2.2.6 Limitations of Current Simulation Methodologies

Almost all the two- and three-dimensional geometric simulations of microstructures (as well as other types of simulations) utilize idealized simple particle shapes such as circles and ellipses in two-dimensions, and spheres and ellipsoids in three-dimensions. Few simulations have been reported in the literature where arbitrary digitized particle shapes have been utilized [8]. But these simulated particle shapes are unrealistic because they are not based on the particle shapes in corresponding real microstructures. This is a major limitation because the particle shapes real material microstructures are often quite complex, and such particle morphologies/shapes do affect the material properties. Recently, a novel digital image processing based technique has been developed at Georgia Tech that can incorporate complex realistic particle shapes and

morphologies in 2D microstructural simulations [91]. There is a need to further develop and extend this simulation technique for incorporation of complex realistic 3D particle shapes and morphologies in simulations of 3D microstructures.

Most of the simulation techniques reported in the literature either assume completely isotropic random orientations of the particles/features or assume that all particles/features have the same orientation (for example, uniaxial aligned fiber composites). On the other hand, the real microstructures are often partially anisotropic having characteristic morphological orientation distribution of the interface normals. Such partial anisotropy influences the material properties as it induces anisotropy in the material properties. Therefore, for simulations of realistic microstructures, there is a need to develop the techniques that simulate the microstructures having specified morphological orientation distribution.

For a simulated microstructure to be realistic, it is essential that the mathematical representation (for example, two-point correlation function) of the simulated microstructure matches closely with the mathematical representation of the corresponding real microstructure. Few geometric simulations have been reported in the literature [7], where the simulated microstructure has an arbitrary specified mean two-point correlation function. However, these techniques lead to microstructure simulations that are (i) two-dimensional, (ii) have isotropic random feature orientations, and (iii) of very small size (typically 512×512 pixels) and therefore, cannot incorporate long-range spatial patterns. Again, in these simulations, either simplified particle shapes are assumed, or arbitrary unrealistic digitized particle shapes are incorporated. Further, these

simulations are not realistic because the target two-point correlation functions were not obtained from experimental data on the real microstructures.

In summary, there is a need to develop techniques for simulations of realistic complex 3D microstructures that (1) incorporate realistic complex particle/feature shapes, (2) allow controlled non-uniformities/clustering in spatial distributions of features, (3) permit partial anisotropic morphological orientations of microstructural features, (4) closely match experimentally measured attributes (spatial correlation functions, orientation distributions, size and shape distributions, volume fraction, etc.) of the corresponding real microstructures, and (5) efficiently generate sufficiently large segments of microstructure that contain short-range (on the order of particle/feature size), intermediate-range (five to ten times particle/feature size), and long-range (few hundred times the particle/feature size) microstructural heterogeneities and spatial patterns. The present research utilizes a combination of digital image processing techniques and computer simulations to develop an efficient and general methodology for representation and simulations of such realistic 3D microstructures.

CHAPTER 3

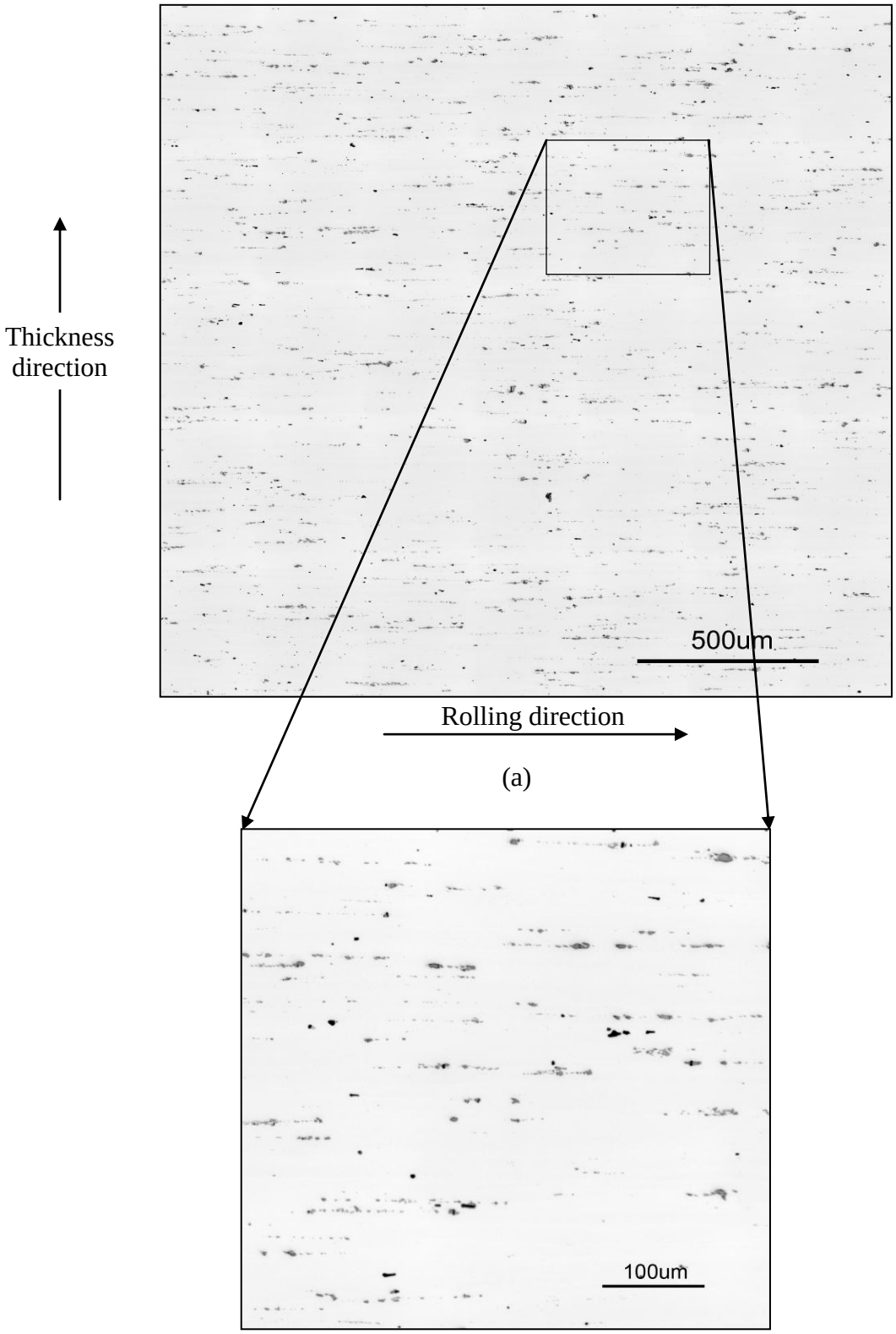
REALISTIC 2D MICROSTRUCTURE SIMULATION

3.1 Introduction

The primary objective of the present research is to develop efficient and flexible techniques for geometric computer simulations of realistic 3D microstructures of heterogeneous materials and apply these techniques for simulations of microstructures of typical materials where particle shapes/morphologies are complex, and significant long-range spatial clustering and/or morphological partial anisotropy exists. For this purpose, preliminary research has been carried out on the microstructure of coarse constituent particles in a commercial hot-rolled Al-Zn-Mg-Cu base 7075 alloy, which is an important Al-alloy of the 7xxx series widely used for aerospace structural applications. This preliminary research on simulation of realistic microstructures observed in 2D metallographic planes through the 3D microstructure of 7075 Al-alloy demonstrates the feasibility of the present research on the simulations of realistic 3D microstructures. The next section of the chapter describes the material, which is followed by the experimental procedures required for microstructure characterization and representation. The subsequent section presents application of a digital image analysis based technique for incorporation of realistic complex morphologies of the particles in the simulated microstructures. These simulated microstructures are presented in the last section of the chapter.

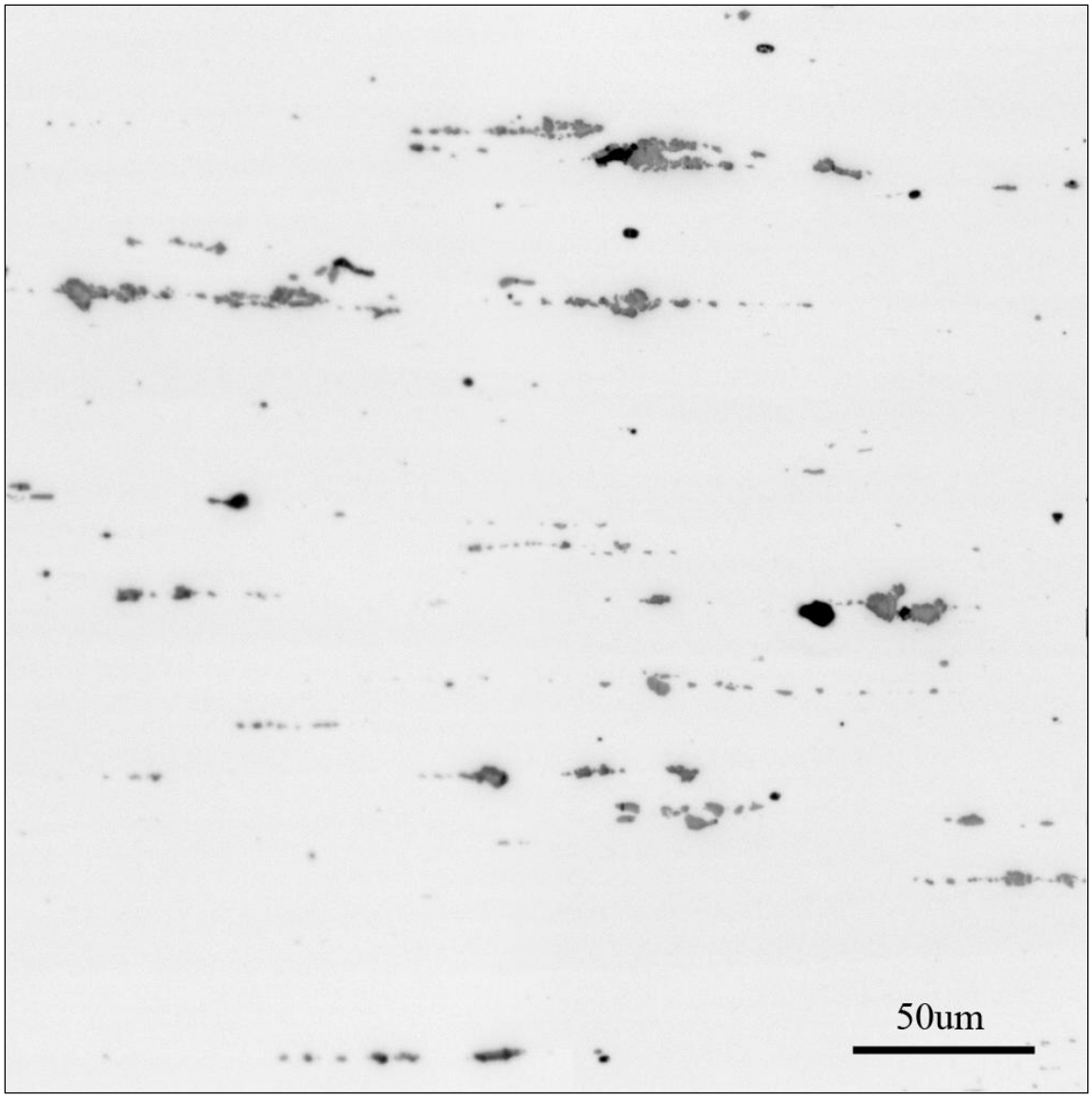
3.2 Material and Microstructural Observations

The experiments have been performed on commercial hot-rolled Al-Zn-Mg-Cu base 7075 T6 Al-alloy. The alloy contains brittle coarse constituent particles or inclusions typically in the size range of 1 to 50 μm. These constituent particles consist of Fe-rich intermetallic compounds and Mg_2Si phase. The constituent particles have complex shapes/morphologies, they are spatially non-uniform (clustered), and they have partially anisotropic morphological orientations (see Figure 3.1). The spatial clustering of the constituent particles is remnant of the cast ingot microstructure, whereas the directionality and morphological anisotropy is primarily due to the hot-rolling process. It is well known that the microstructural geometry of the constituent particles significantly affects fracture toughness [92-93], fatigue resistance [94], damage evolution [95] and corrosion resistance [96], as well as anodizing behavior [97] and recrystallization behavior [98] of hot-rolled Al-alloys. Therefore, it is of interest to incorporate *realistic* complex shapes/morphologies, non-uniform spatial arrangements, and morphological orientation distribution functions of the coarse constituent particles in the microstructure models and simulations, which can be subsequently implemented in the computational models and simulations of the mechanical behavior and processing of wrought Al-alloys.



(a)

(b)



(c)

Figure 3.1: Microstructure ofhot rolled Al-Zn-Mg-Cu base 7075 alloy in L-S plane
(a) Montage of 64 contiguous microstructural fields,
(b) Region containing 4 contiguous microstructural fields,
(c) One field of view.

3.2.1 Metallography

Metallography of 7075 T6 alloy was performed by J. Harris on the material cut from the grip section of a standard tensile test specimen; the details are reported in his M.S. thesis [99]. In these tensile tests, the applied tensile stress was along the direction parallel to the rolling direction. The tests were performed at room temperature in the strain-controlled mode at the strain rate of 10^{-4}/s. The microstructure of the specimen failed at 15% engineering strain has been utilized in the present work. The metallographic specimens were cut along (i) an L-S plane containing the rolling direction (L) and the thickness (or short transverse) direction (S), (ii) L-T plane containing the rolling direction (L) and transverse (T) direction, and (iii) S-T plane of the rolled plate. These specimens were mounted in standard metallographic cold mounts, polished using a series of SiC polishing papers (240 to 600 grit size) followed by diamond pastes (6 μm to 2 μm), and finally polished using colloidal silica. The microstructure was observed and characterized in unetched condition.

3.2.2 Digital Image Analysis

Digitally compressed montages of 64 (8 × 8) contiguous microstructural fields were generated using image processing methods developed by Louis and Gokhale [2, 4]. Individual contiguous microstructural fields of 1000 × 1000 pixels at a pixel size of 0.32 μm were grabbed using an automated AxioVision image analysis system connected to a Zeiss Axioscope microscope, stored in the computer memory, and subsequently joined seamlessly using image processing to produce a seamless large montage of 64 contiguous microstructural fields (6300 × 6300 pixels). Small constituent particles (< 1 μm size) were scrapped from the montages because their attributes could not be accurately

measured. Some of the very small particles are the artifacts due to the noise in the digital images, and small particles do not adversely affect the mechanical behavior of the alloy. Figure 3.1 depicts the microstructure of the L-S metallographic plane revealed in this manner at three length scales. Figure 3.1a is a digitally compressed montage of 64 contiguous microstructural fields that covers an area of 4 mm^2 and contains about 2600 constituent particles, which shows long-range clustering and spatial patterns of the constituent particles. Figure 3.1b shows a smaller segment of the montage in Figure 3.1a consisting of 4 contiguous microstructural that reveals banding and anisotropy of the constituent particles, and Figure 3.1c is one field of view of the montage in Figure 3.1a. All the microstructural fields in Figure 3.1a have the same resolution as that depicted in Figure 3.1c. Large number of microstructural montages (such as Figure 3.1a) were grabbed in the three metallographic planes of interest for reliable quantitative microstructure characterization. The montage construction is essential to quantify and represent short-range (on the order of particle/feature size), intermediate-range (five to ten times particle/feature size), and long-range (few hundred times the particle/feature size) microstructural heterogeneities and spatial patterns.

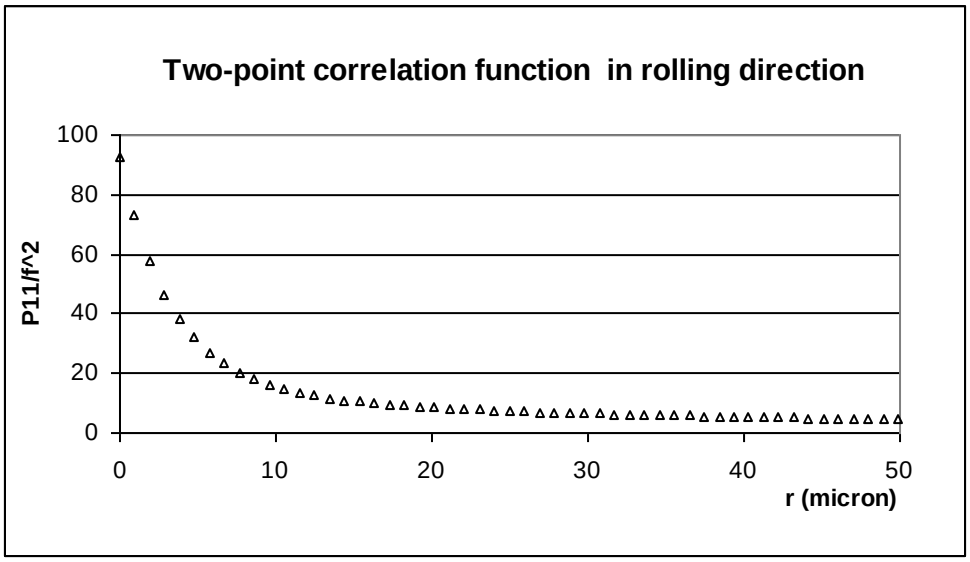
3.2.3 Quantitative Microstructure Characterization and Representation

For statistically reliable quantitative microstructure characterization, the microstructure sample size must be such that any two microstructure samples at different locations must yield closely matching (say within 5%) microstructural attributes. In this context, the present microstructure is extremely heterogeneous: even one large montage such as that Figure 3.1a cannot be regarded as a representative microstructural segment because two such montages at different locations in the same plane do not necessarily

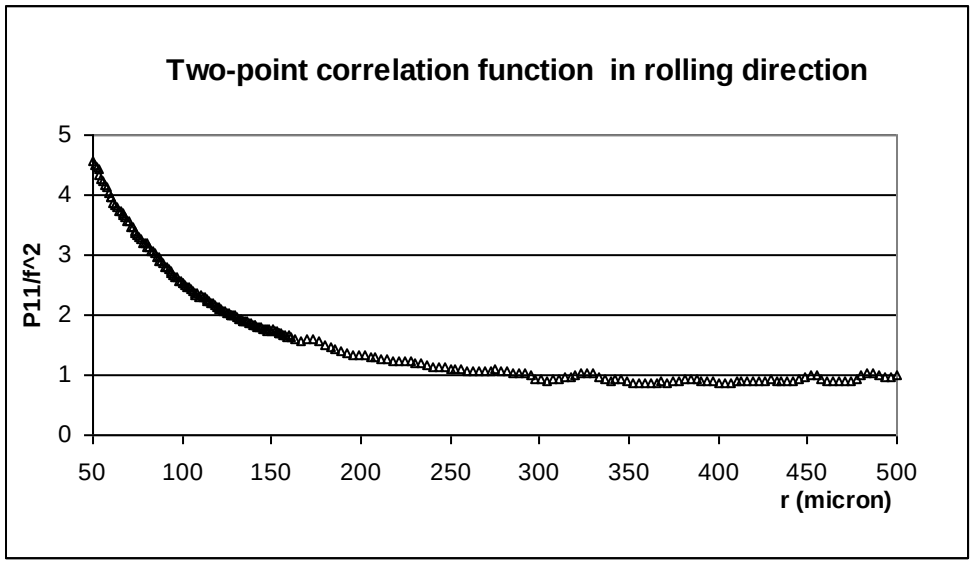
have the same constituent particle area fraction, two-point correlation function, etc. For the present microstructure, by trial and error, it has been found that for reliable microstructure characterization, the microstructural sample size should be 20 montages of the size shown in Figure 3.1a captured at different uniform random locations in the specimen. The microstructural attributes averaged over two independent sets of such 20 montages are found to be within 5% of one another. Therefore, the experimental two-point correlation functions reported in this chapter have been obtained by averaging the data over 20 microstructural montages, each containing 64 contiguous microstructural fields. Consequently, the present experimental two-point correlation data in each metallographic plane involved measurements on microstructure sample that contained about 52,000 constituent particles.

In the present microstructure, the spatial clustering of the constituent particles is remnant of the cast ingot microstructure, whereas directionality and morphological anisotropy are primarily due to the hot-rolling process. Therefore, it is convenient to describe the direction dependence of this microstructure with respect to the rolling direction. Consequently, the rolling direction is designated as the Z-axis, and for convenience, the thickness direction of the plate is designated as the X-axis of the XYZ frame of reference. As a result, for the two-point correlation function $P_{11}(r, \theta, \phi)$, θ is the angle between the line of length r and the Z-axis (rolling direction). On the other hand, ϕ represents the rotation of the line around the rolling direction, which is equal to the angle between the projection of the line on the XY plane and the X-axis (thickness direction). As a result, for line directions in the L-S metallographic plane (containing the rolling direction and thickness direction), ϕ is equal to zero, and θ varies from 0 to π. For the

lines parallel to the rolling direction, θ is equal to zero, whereas it is equal to $\pi/2$ for lines parallel to the thickness direction of the rolled plate. For the directions in the L-T metallographic plane (containing the rolling direction and thickness direction), ϕ is equal to $\pi/2$, and θ varies from 0 to π, whereas in the S-T plane, θ is equal to $\pi/2$ and ϕ varies from 0 to π. The two-point correlation functions were measured in the L-S, L-T, and S-T metallographic planes at different physically meaningful values of r, θ, and ϕ using the digital image analysis based procedure described elsewhere [68]. Figure 3.2 depicts a two-point correlation function in the L-S plane obtained in this manner. Note that, in this plot, P_{11} (r, θ, 0) is normalized by the square of the particle volume fraction (f^2), and therefore, it is expected to approach the value of 1.0 at very large values of r. To obtain representative microstructural data, each two-point correlation function is obtained by averaging the data over 20 montages, each of area 6300 × 6300 pixels having pixel size of 0.32 μm. Each montage contained approximately 2600 particles. Each two-point function data point is obtained from about 800 million automated computational observations**.** Therefore, the data set is robust and representative of the microstructure. The two-point correlation functions along the other directions in the L-S plane as well as along the directions in L-T and S-T planes have also been measured using the same procedure. Figures 3.3 and 3.4 show two-point correlation functions measured in L-T and S-T planes. The computer code for measuring two-point function is given in Appendix B.1. In the next section, these experimental data are used to simulate the microstructures that have statistically similar two-point correlation functions, similar complex particle morphologies, and the same first order microstructural attributes such as particle volume fraction, number density, and size distribution.

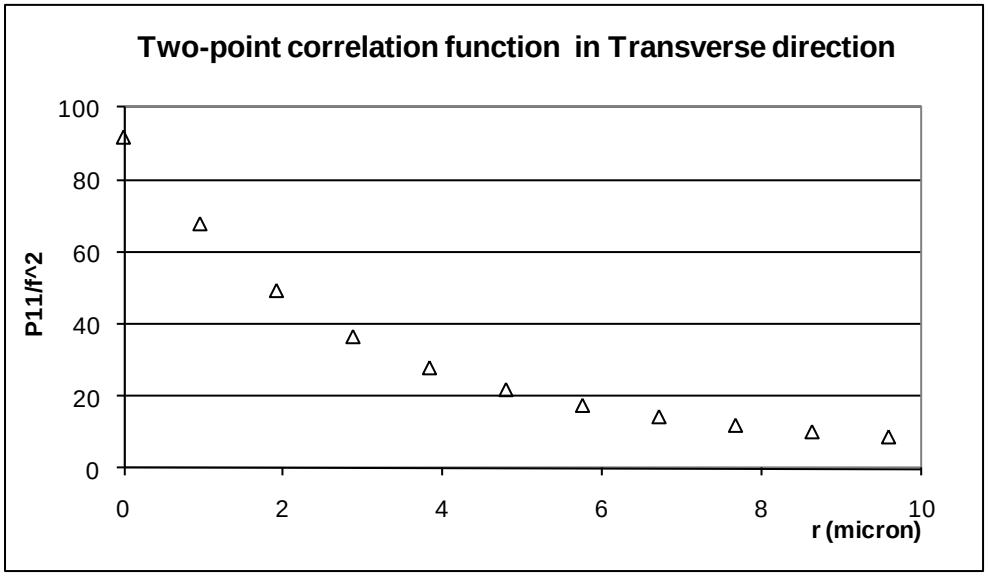


(i)

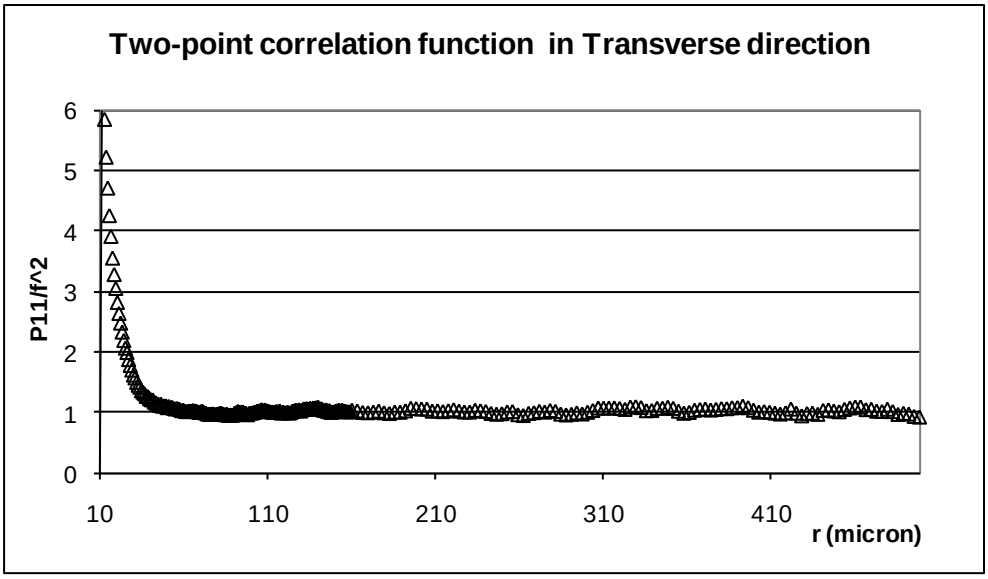


(ii)

Figure 3.2: Two-point correlation function of the constituent particles along the rolling direction in L-S plane, normalized by their respective volume fraction squares, averaged over 20 montages. (i) Short range data set, (ii) long range data set.

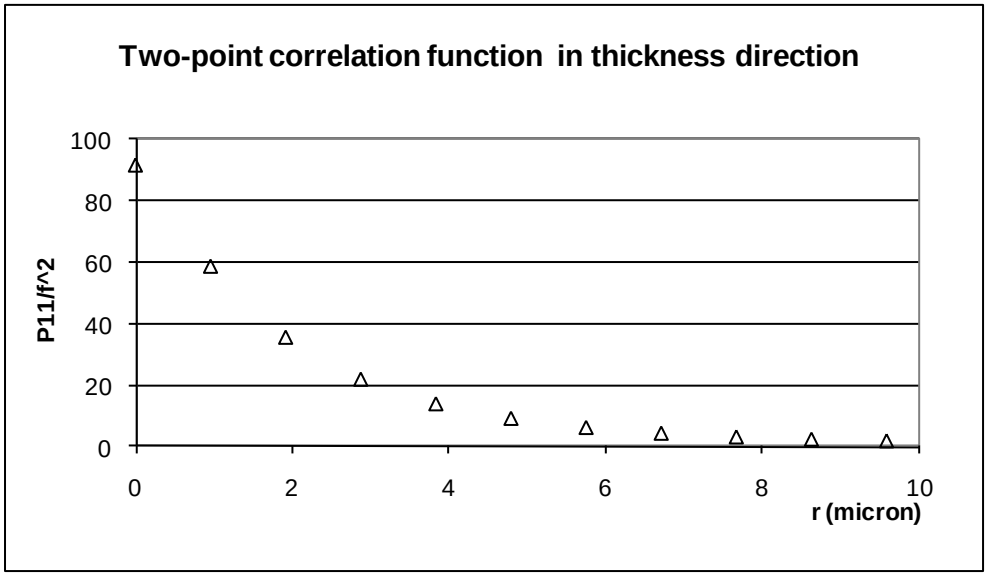


(i)

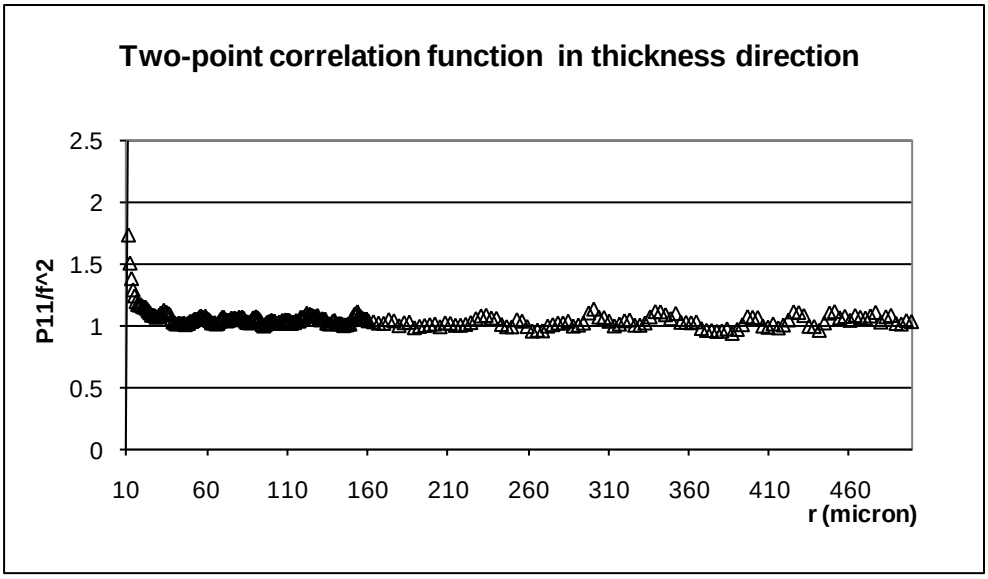


(ii)

Figure 3.3: Two-point correlation function of the constituent particles along the transverse direction in L-T plane, normalized by their respective volume fraction squares, averaged over 20 montages. (i) Short range data set, (ii) long range data set.



(i)



(ii)

Figure 3.4: Two-point correlation function of the constituent particles along the thickness direction in S-T plane, normalized by their respective volume fraction squares, averaged over 20 montages. (i) Short range data set, (ii) long range data set.

3.3 Computer Simulation of Realistic 2D Microstructure

In the present work, a combination of digital image processing and microstructure simulation algorithms is used for the simulations of realistic microstructures. Conceptually, the simulation procedure is as follows. Focus on the high magnification high-resolution microstructural images such as the one in Figure 3.1c. Such microstructural images obviously contain the particle section images having real morphologies. Now, consider a thought experiment where sufficiently large number of constituent particle images from such microstructural fields of view are simply "plucked" out and stored in a box, such that the set is representative of the size, shape, and morphology distribution of the entire particle population in the microstructure. Next, consider simulation of constituent particle centroids (as per some specified spatial arrangement and number density) in a digitized simulation space where the pixel size is the same as that in the microstructural images from which the particles are plucked out. Finally, thoroughly "shake" the box containing the plucked out constituent particle images, take out one particle image at random, and place it at one simulated centroid. Specify all pixels in the particle as binary dark pixels. Repeat the process until there is one particle image centered on each simulated centroid. The result is a simulated microstructure containing the same constituent particle morphologies and size distribution as those in the real microstructural images but a different spatial arrangement of the particles. Finally, compute the two-point correlation functions along different directions in the simulated microstructure, compare those with the corresponding experimentally measured two-point functions, and vary the simulated microstructure (via change of simulation parameters) until a satisfactory match between experimental and

simulated two-point correlation functions is achieved. The process then yields a simulated microstructure in the corresponding metallographic plane (L-S or L-T, or S-T) of the hot-rolled plate that is statistically similar to the corresponding real microstructure in that plane. Note that, if necessary, higher order correlation functions can also be matched in the similar manner, or descriptors other than correlation functions (for example, radial distribution function) can also be used. The important steps involved in this methodology are described in detail as follows.

3.3.1 Capturing Real Particle Morphologies

The set of (*X, Y*) coordinates of closely spaced points (pixels) on the boundary of the binary image of a particle contains complete detailed information on the morphology and geometry of that particle. Once such a set of boundary/contour points is available, the exact replica of that particle can be then reproduced at any desired location in the simulation space. Ren, Yang, and Sun [100]] have given an image analysis procedure to extract the boundary contours of the features in a binary digital image, which can yield the (*X, Y*) coordinates (pixel positions, to be more precise) of the closely spaced points (pixels) on the boundary of the feature. An in-house C++ computer code developed for this research uses digital binary images of the microstructure. The code identifies pixels on the boundaries of a particle and creates "inner" and "outer" boundary contours. A boundary contour is classified as an outer contour if it encloses the particle, and it is considered as an inner contour if it encloses the matrix and is surrounded by the particle. Extraction of the coordinates/positions of the boundary pixels is a three-step process. First, a starting point is found on a boundary, and then the contour is followed pixel by pixel, and finally, the termination of the contour is identified. In this way, the code

generates the set of coordinates/locations of the pixels that form the boundary of a particle. Using this image analysis procedure, the boundary contours of a few thousand constituent particles were extracted to represent the size and shape distribution of the particle population observed metallographically. The data set contains the positions of the boundary pixels of each particle. The centroid pixel position of each particle is then computed from the positions of the boundary pixels. Using simple coordinate translation (change of origin), the (X, Y) positions of the pixels on the boundary contour are changed so that each particle centroid is at a (0, 0) position. The data set corresponding to each particle is then stored in the computer memory. Next, each of N particles is assigned a distinctive number in the range of 1 to N to identify that particle; these identification numbers are assigned in a random manner, and they have no correlation to the size or shape of the particles. The computer code is given in Appendix B.3.

3.3.2 Simulation of Particle Rich and Particle Poor Regions in the Simulation Space

In the present alloy, the constituent particles are spatially clustered because during the ingot solidification, the constituent particles are formed in the inter-dendritic regions of the cast ingot. The hot-rolling of the ingot leads to deformation of the particles into elongated morphologies, anisotropic orientations, and transformation of the particle clusters in the inter-dendritic regions into anisotropic bands of particles that are mostly aligned parallel to the rolling direction of the plate. Nonetheless, as the deformation due to the rolling process is not necessarily uniform at all locations, some constituent particle clusters have low aspect ratio and they are relatively less elongated. Therefore, three types of cluster bands (representing constituent particle rich regions) are first simulated, namely, the high aspect ratio, intermediate aspect ratio and low aspect ratio bands. The

number densities of these three types of regions, their size, orientations and aspect ratios are important simulation parameters. These bands are simulated using the well-known RSA algorithm [25]. The bands are not permitted to overlap.

3.3.3 Simulation of Constituent Particle Centroids in the Particle-Rich and Particle-Poor Regions

Within each band, constituent particle centroids are simulated at uniform random locations with specified number density $[N_A]_{cluster}$. The constituent particle centers are also simulated at uniform random locations at all other locations of the simulation space not covered by the clustered regions but at a lower number density $[N_A]_{poor}$. The clustering intensity of the constituent particle is then given by the parameter $[N_A]_{cluster}/[N_A]_{global}$, where $[N_A]_{global}$ is the global average number density of the constituent particles in the simulated microstructure. The "skeleton" of the simulated microstructure is created in this manner; placing constituent particles at these simulated particle centroids then generates the simulated microstructure. The computer code of this part of simulation is given in Appendix B.4.

3.3.4 Placement of Constituent Particles at the Simulated Centroids

The constituent particles are sequentially placed at different simulated centroids using their ID numbers. The placement of a particle at a location in the simulation space simply involves translation of the particle centroid from (0, 0) to the pixel coordinates of the new location in the simulation space. In this manner a constituent particle is placed at each simulated centroid. In the present simulations, the particle overlaps are not permitted, which is similar to the RSA algorithm. However, the computer code is flexible enough to permit the constituent particles to freely overlap (as in Boolean schemes [27]),

or to permit limited overlaps (similar to the cherry-pit model [87]), if needed for other types of microstructures. The computer code of this part of simulation is given in Appendix B.5.

3.3.5 Comparison of Two-Point Correlation Functions of Simulated and Real Microstructures

For a chosen size and shape distribution of constituent particles, the simulation parameters that can be changed to vary the microstructure are as follows.

- Number densities of three types of particle rich bands
- Size, aspect ratios, and orientations of the particle rich bands
- Volume fraction of constituent particles
- Clustering intensity, i.e., $([N_A]_{cluster}/[N_A]_{global})$

In the present technique, it is necessary to begin with a set of "guess" values for the above simulation parameters. A microstructure is then simulated with that combination of the simulation parameters, and its two-point correlation function is computed for different values of *r* ranging from 1 μm to 500 μm and for different line orientations in that plane. As the real microstructure of the constituent particles is highly heterogeneous, as expected, the corresponding simulated microstructure is also very heterogeneous. Consequently, two large simulated microstructural windows of the size of the montage shown in Figure 3.1a are generated by using exactly the same values of the simulation parameters that do not necessarily have exactly the same two-point correlation function. Figure 3.5 shows two-point correlation data for two such microstructural windows that were simulated using the same values of the simulation parameters. For the present microstructure model, it is essential to average the simulated two-point

correlation data over 20 independent simulation montages of 6300 × 6300 pixels to obtain reproducible (within 5% variation) simulated two-point correlation data. Therefore, such averaged two-point function data $[P_{11}(r, \theta, \phi)]_{sim}$ for the simulated microstructures are compared with the corresponding averaged data for the corresponding real microstructure $[P_{11}(r, \theta, \phi)]_{exp}$. At any given value of r, θ, and ϕ, the absolute fractional error $E(r, \theta, \phi)$ can be computed as follows.

$$E(r,\theta,\phi) = \frac{|[P_{11}(r,\theta,\phi)]_{sim} - [P_{11}(r,\theta,\phi)]_{exp}|}{[P_{11}(r,\theta,\phi)]_{exp}} \qquad \text{Eq. 4}$$

Let $\langle E(\theta, \phi)\rangle$ be the average value of $E(r, \theta, \phi)$ averaged over all values of r for a given direction (θ, ϕ). In the present work, a simulated microstructure is considered to be representative of the corresponding real microstructure, if and only if, (1) $\langle E(\theta, \phi)\rangle$ is less than or equal to 0.05 for each direction (θ, ϕ) of interest, and (2) for each value of r, θ, and ϕ, $E(r, \theta, \phi)$ must be less than or equal to 0.10. These maximum errors are acceptable because both the simulated and real microstructures contain "random noise" of this extent. The process of matching simulated and real microstructure involves numerous iterations of simulated microstructures with different combinations of the simulation parameters until the above two conditions are satisfied.

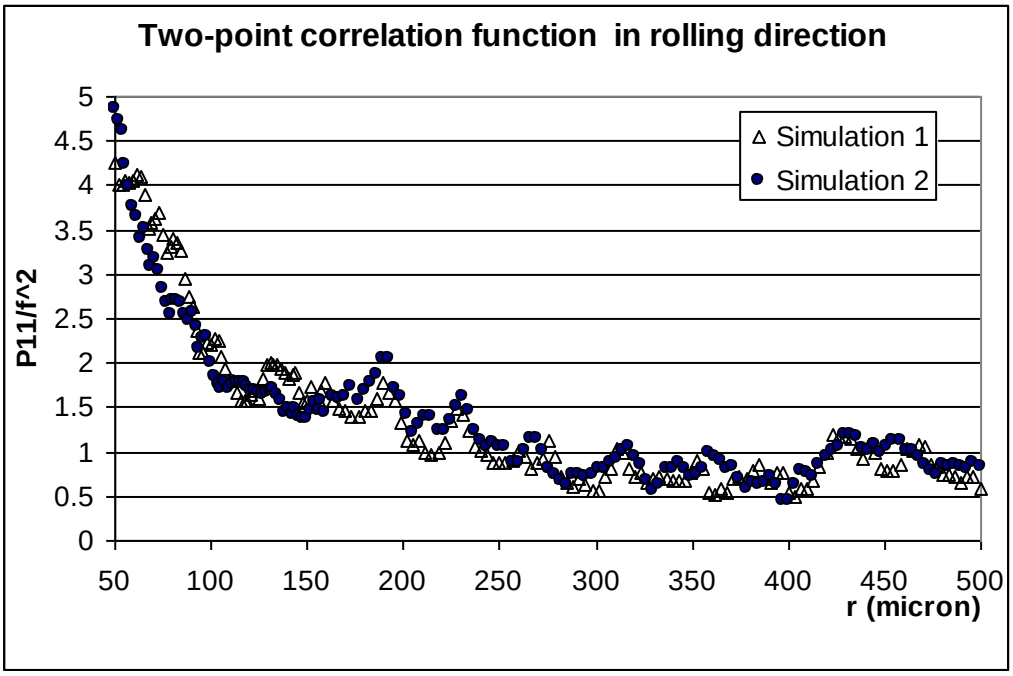
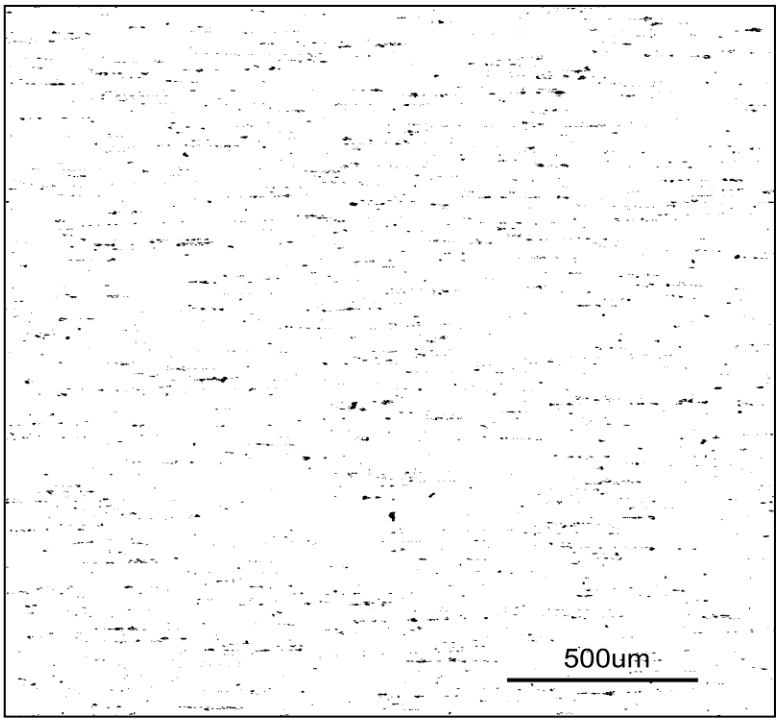


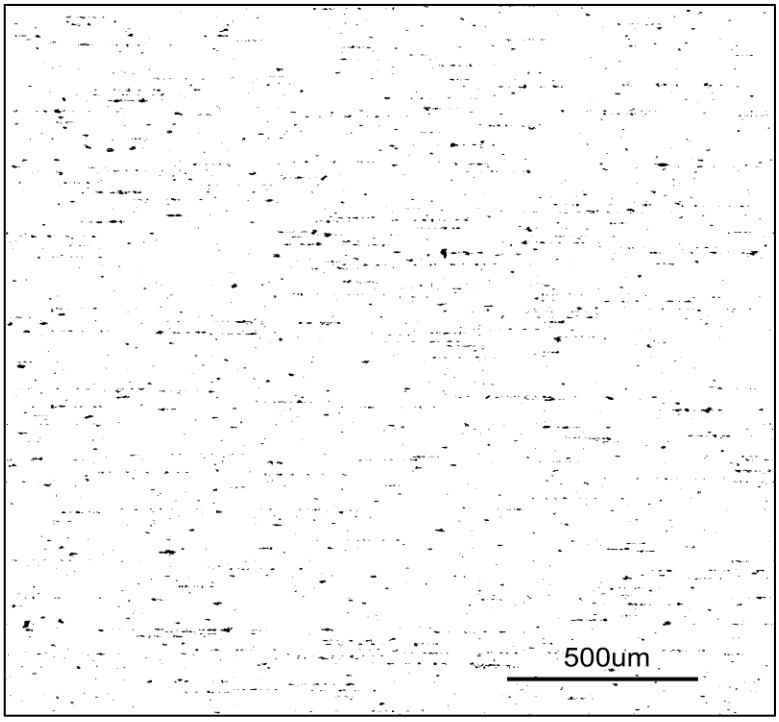
Figure 3.5: Normalized two-point correlation data for two simulations having the same values of the simulation parameters.

3.3.6 Computer Simulations Results

Figures 3.6 and 3.7 depict the simulated microstructure in the L-S plane obtained in this manner corresponding to the real microstructure shown in Figure 3.1. Each simulated image also contains about 2600 particles, and it covers an area of 6300×6300 pixels having pixel size of 0.32 μm. Figures 3.8 to 3.10 compare few experimental and simulated two-point correlation functions of the L-S planes. Figures 3.11 and 3.12 show the simulated microstructures of L-T and S-T planes, and Figure 3.13 shows a perspective of the real and simulated microstructure in the three metallographic planes of interest. Figures 3.14 to 3.19 depict the comparison of a few experimental and simulated two-point correlation functions in L-T and S-T planes.

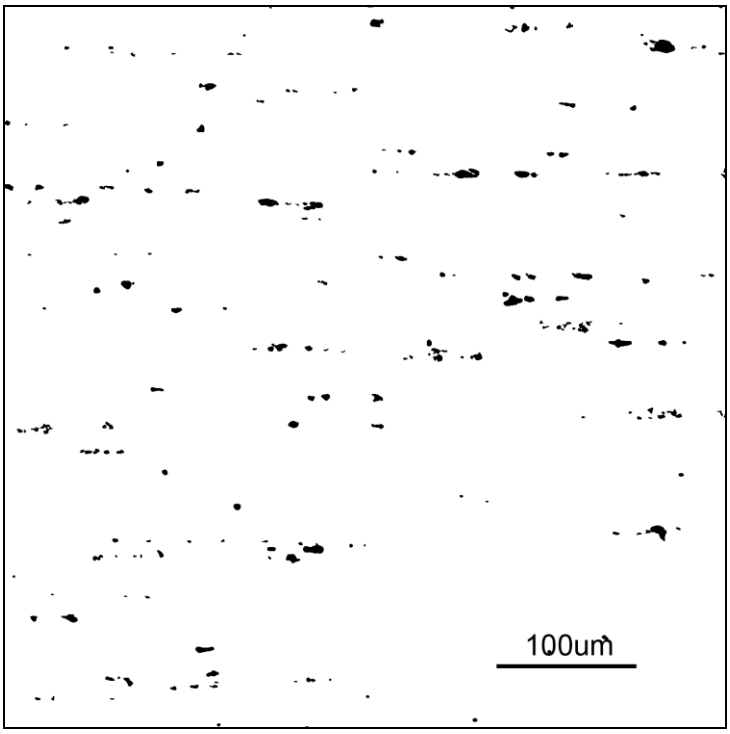


(i)

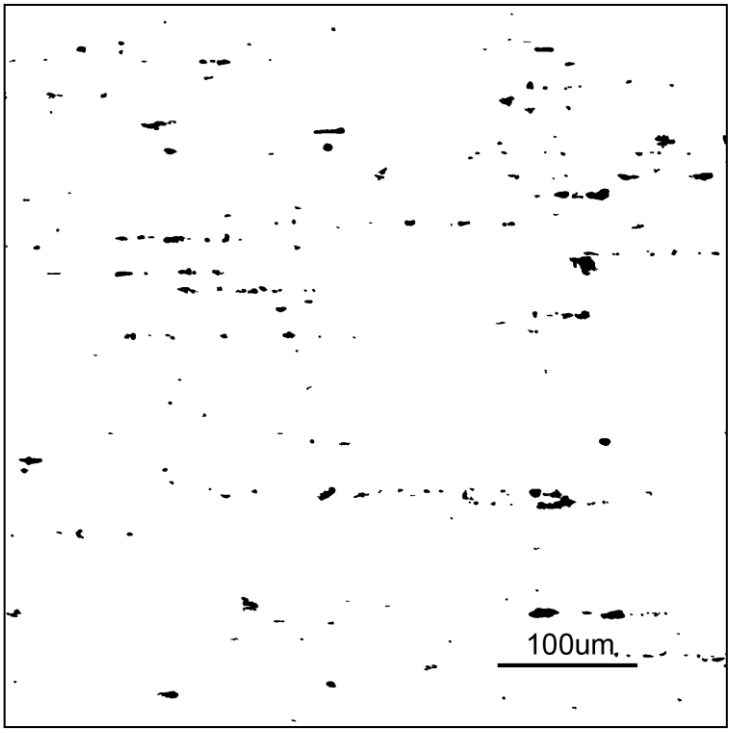


(ii)

Figure 3.6: Comparison of (i) real microstructure and (ii) simulated microstructure in L-S plane

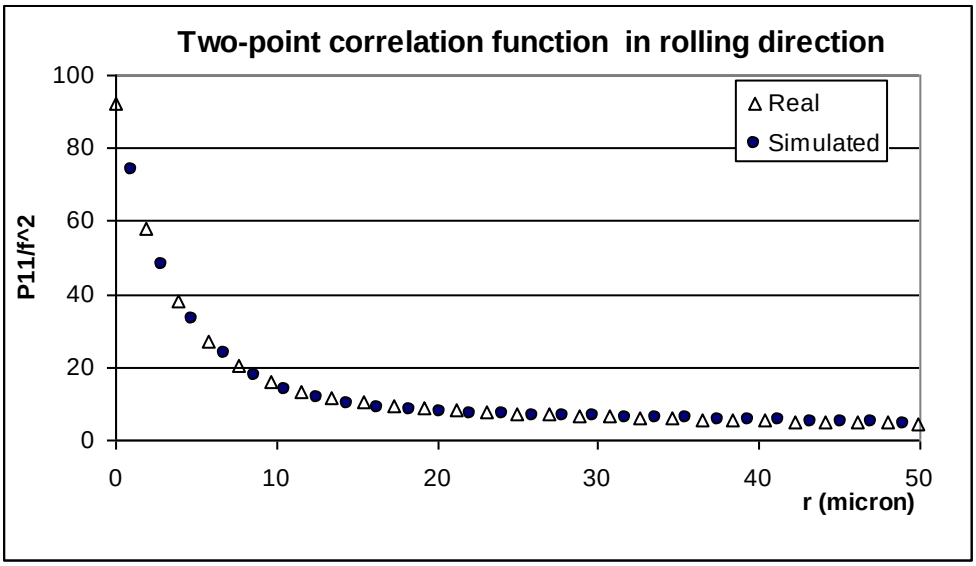


(i)

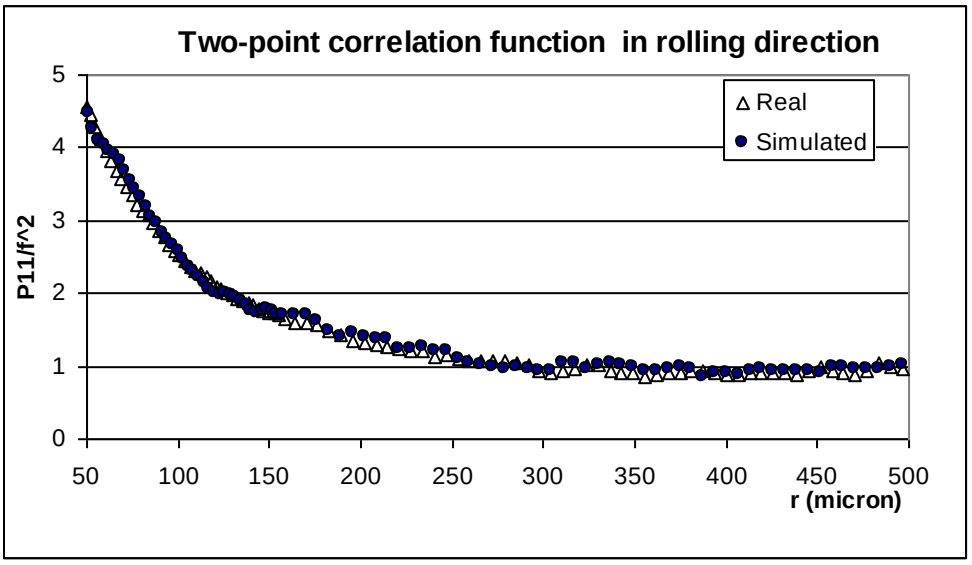


(ii)

Figure 3.7: Magnified view of (i) real microstructure and (ii) simulated microstructure in L-S plane

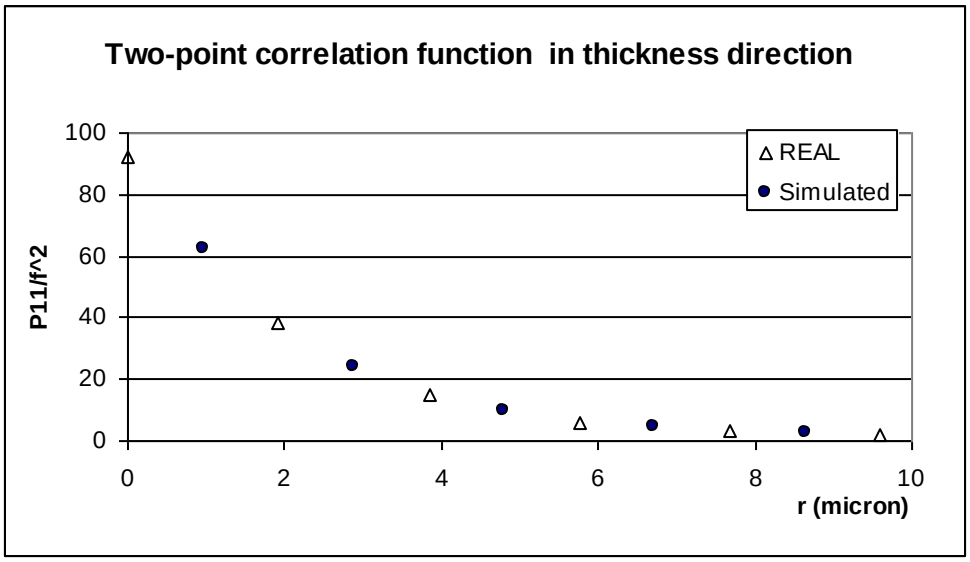


(i)

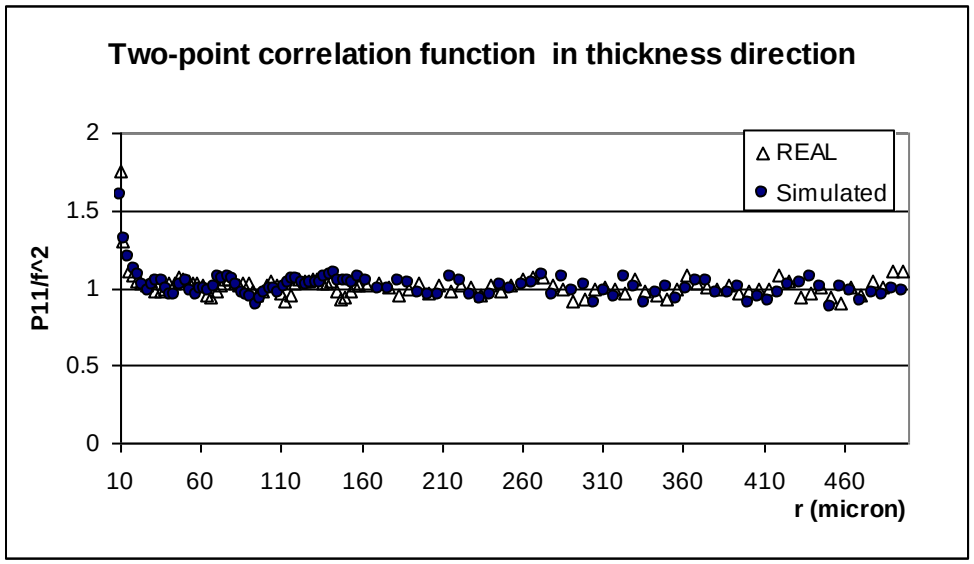


(ii)

Figure 3.8: Comparison of normalized two-point correlation functions of the constituent particles in the L-S plane along the rolling direction in real and simulated microstructures. (i) Short range data set, (ii) long range data set.

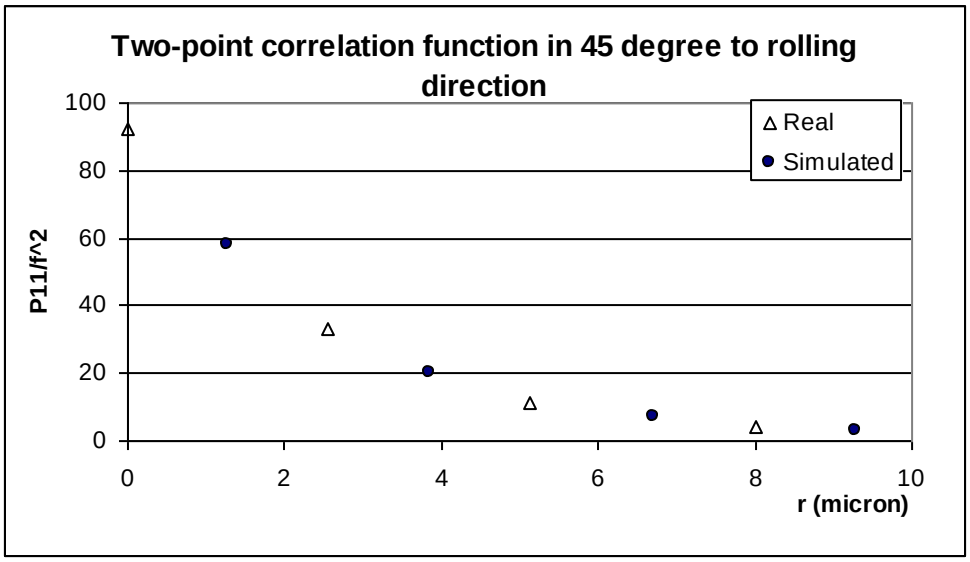


(i)

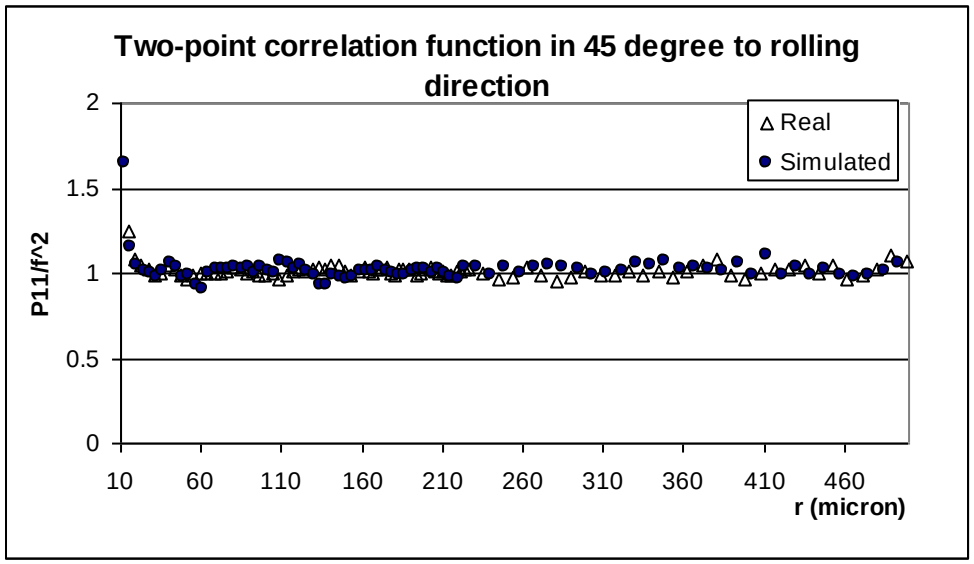


(ii)

Figure 3.9: Comparison of normalized two-point correlation functions of the constituent particles in the L-S plane along the thickness direction in real and simulated microstructures. (i) Short range data set, (ii) long range data set.

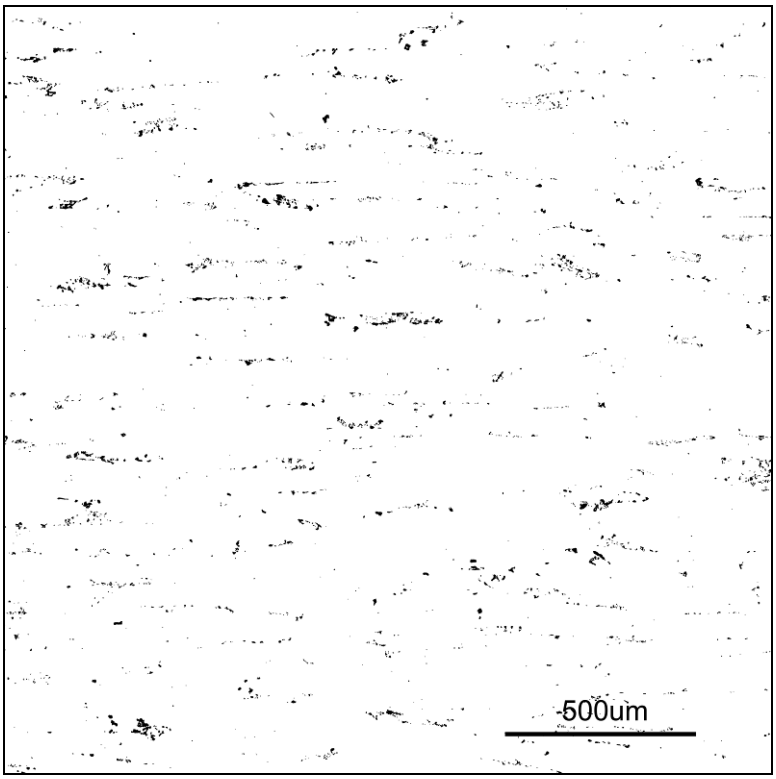


(i)

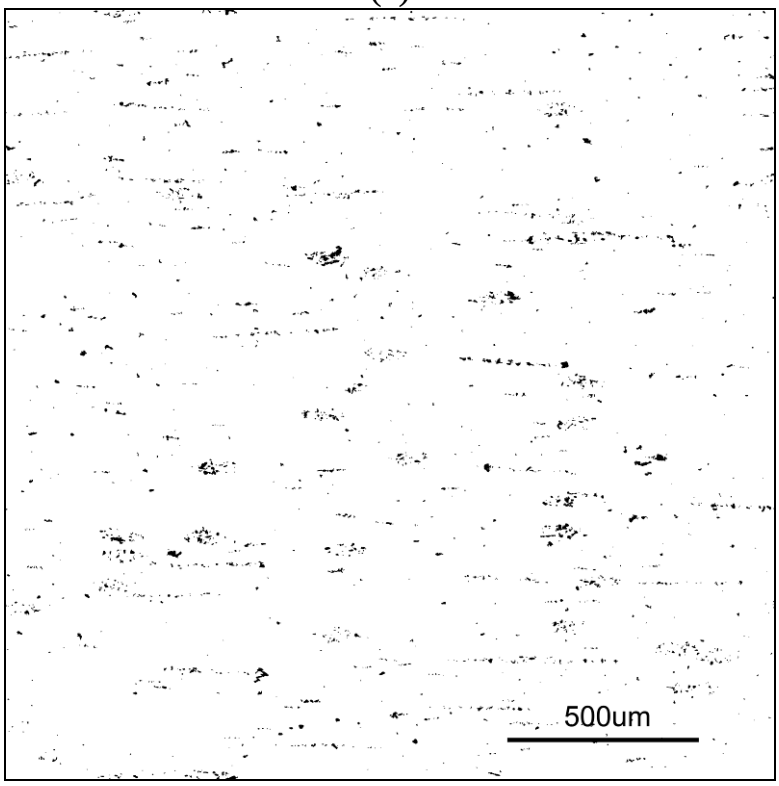


(ii)

Figure 3.10: Comparison of normalized two-point correlation functions of the constituent particles in the L-S plane along 45 degree to rolling direction in real and simulated microstructures. (i) Short range data set, (ii) long range data set.

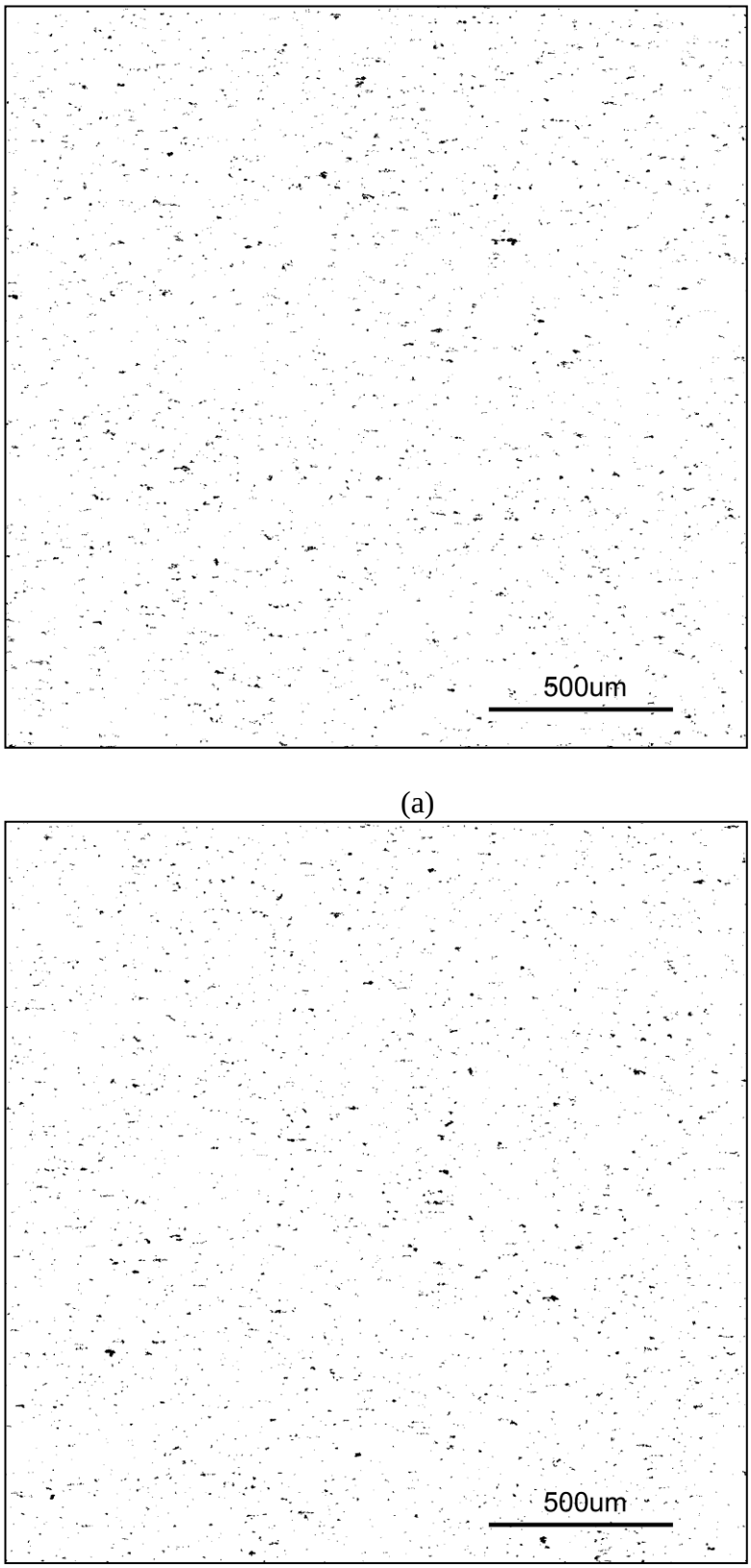


(a)



(b)

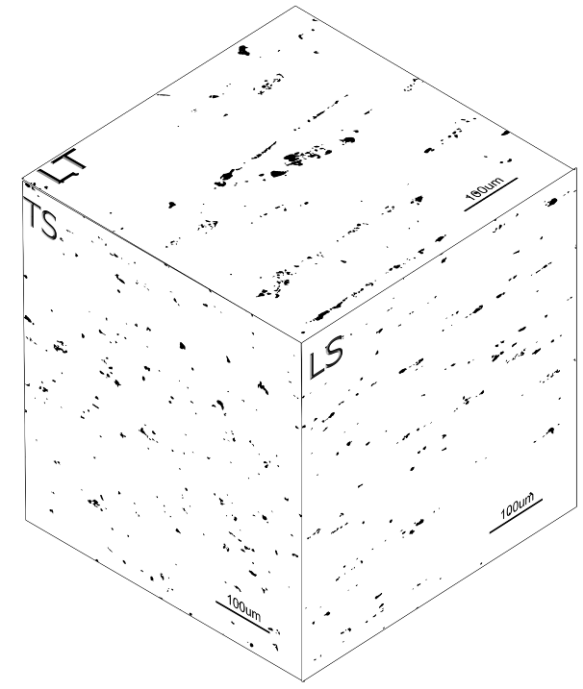
Figure 3.11: Comparison of (a) real microstructure (b) simulated microstructure in L-T plane



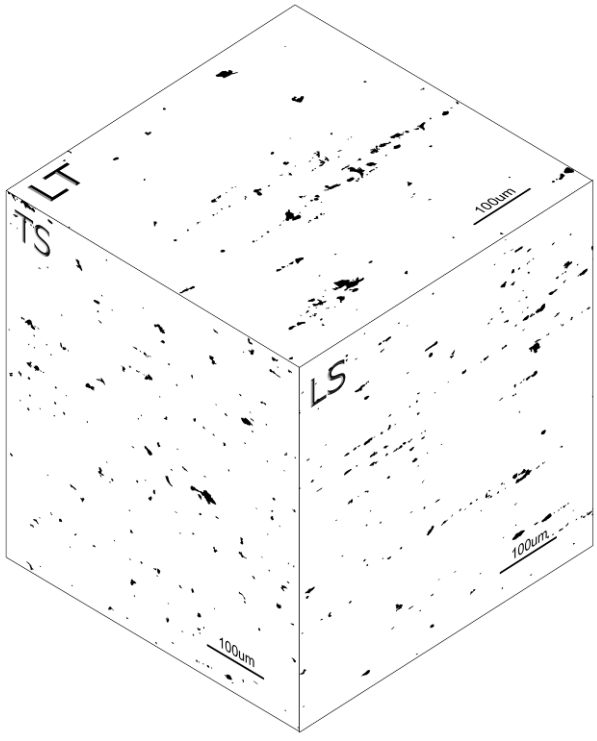
(a)

(b)

Figure 3.12: Comparison of (a) real microstructure (b) simulated microstructure in S-T plane

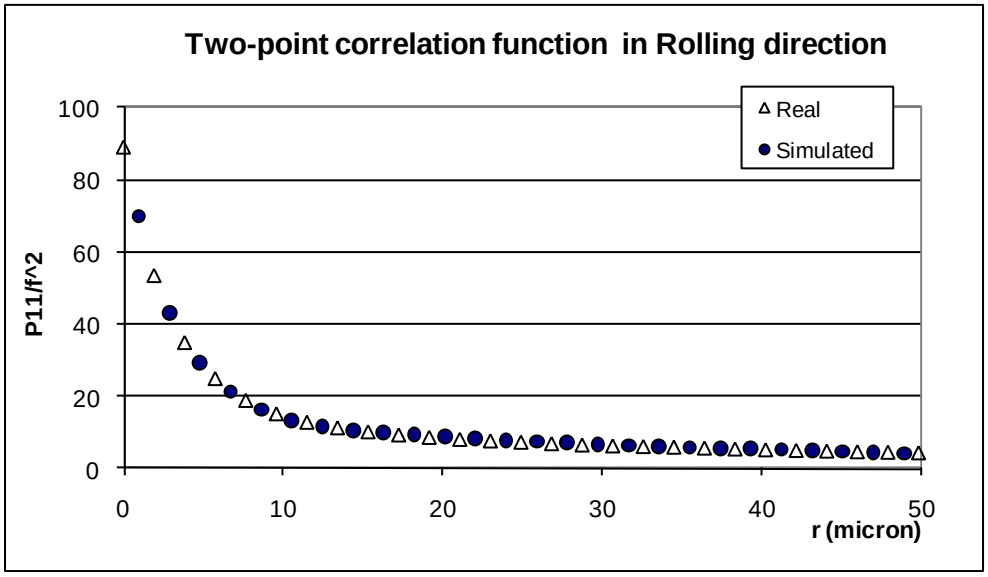


(a)

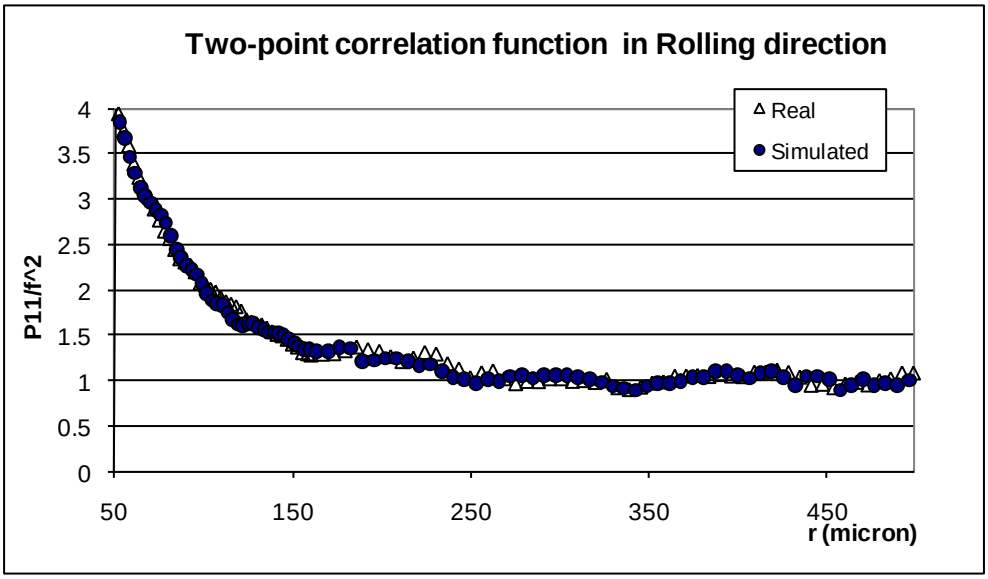


(b)

Figure 3.13: A perspective of (a) real and (b) simulated microstructure in L-S, L-T and S-T plane

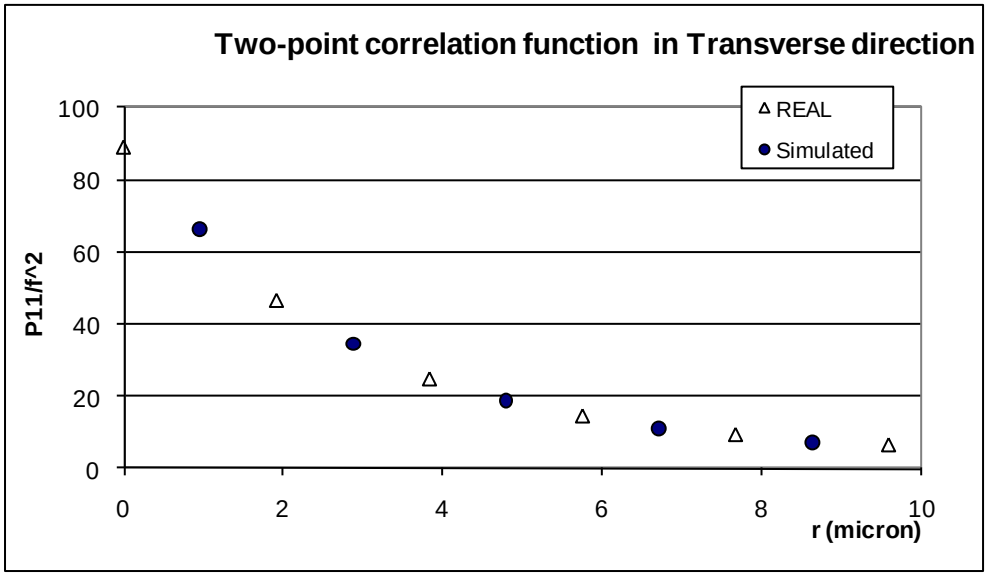


(i)

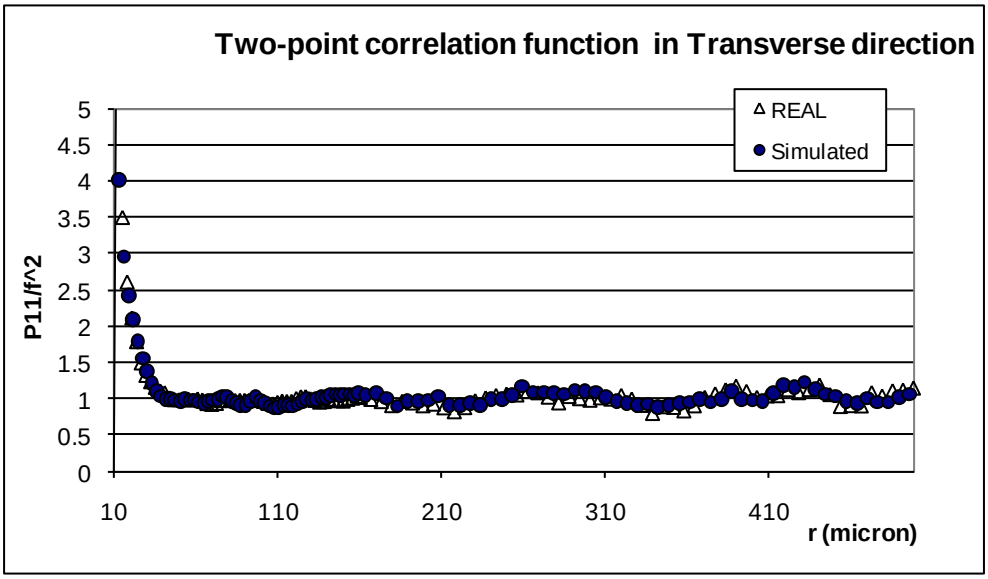


(ii)

Figure 3.14: Comparison of normalized two-point correlation functions of the constituent particles in the L-T plane along the rolling direction in real and simulated microstructures. (i) Short range data set, (ii) long range data set.

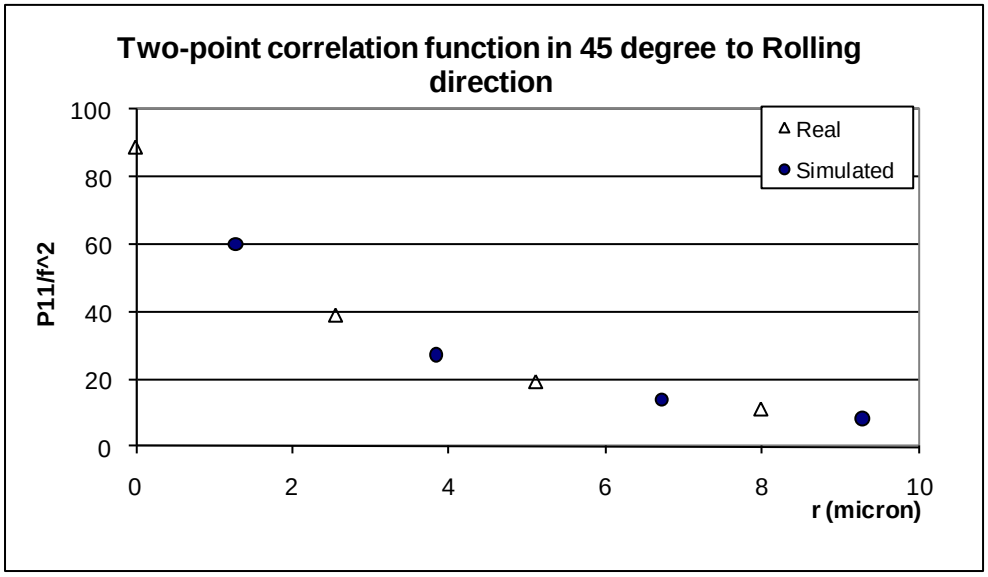


(i)

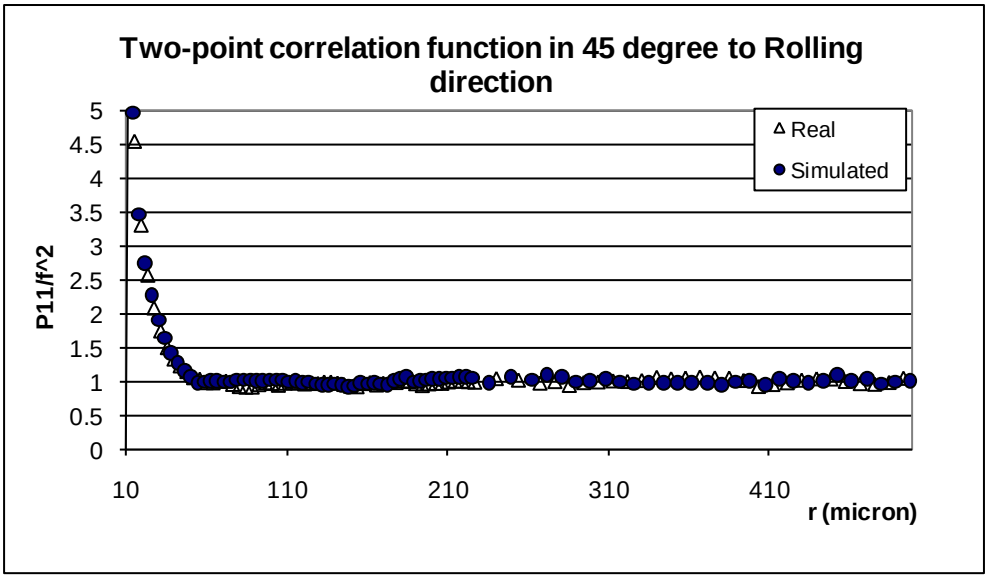


(ii)

Figure 3.15: Comparison of normalized two-point correlation functions of the constituent particles in the L-T plane along the transverse direction in real and simulated microstructures. (i) Short range data set, (ii) long range data set.

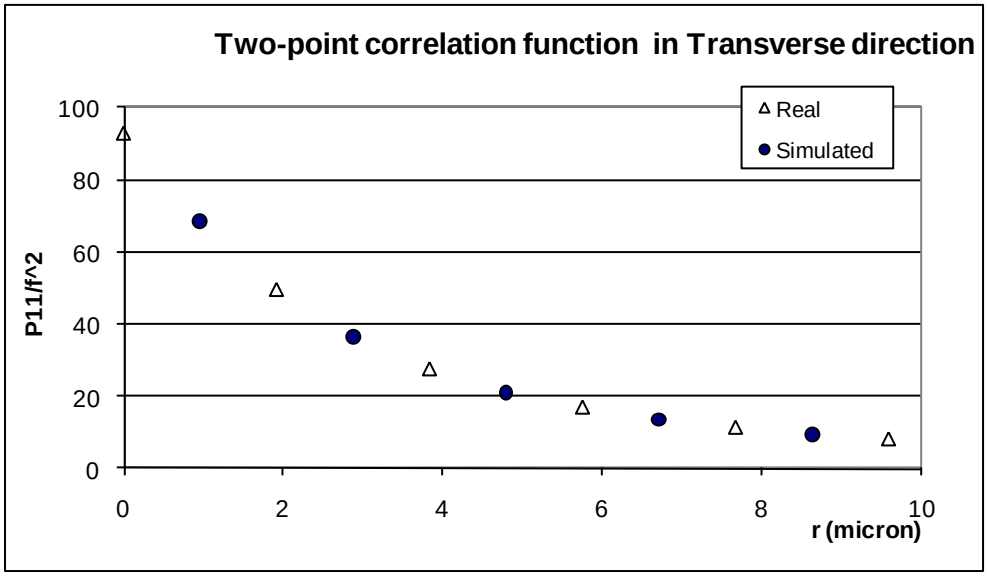


(i)

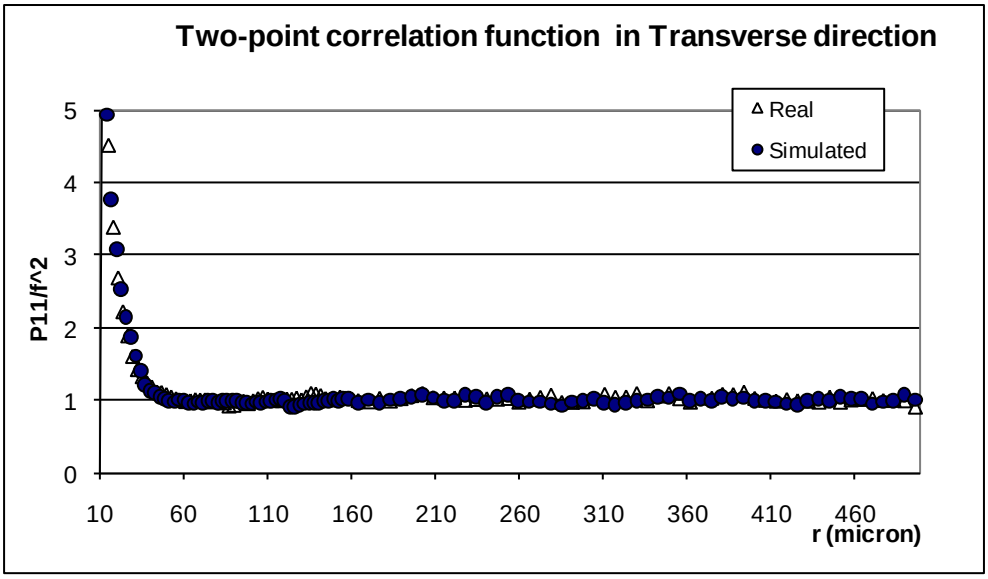


(ii)

Figure 3.16: Comparison of normalized two-point correlation functions of the constituent particles in the L-T plane along 45 degree to rolling direction in real and simulated microstructures. (i) Short range data set, (ii) long range data set.

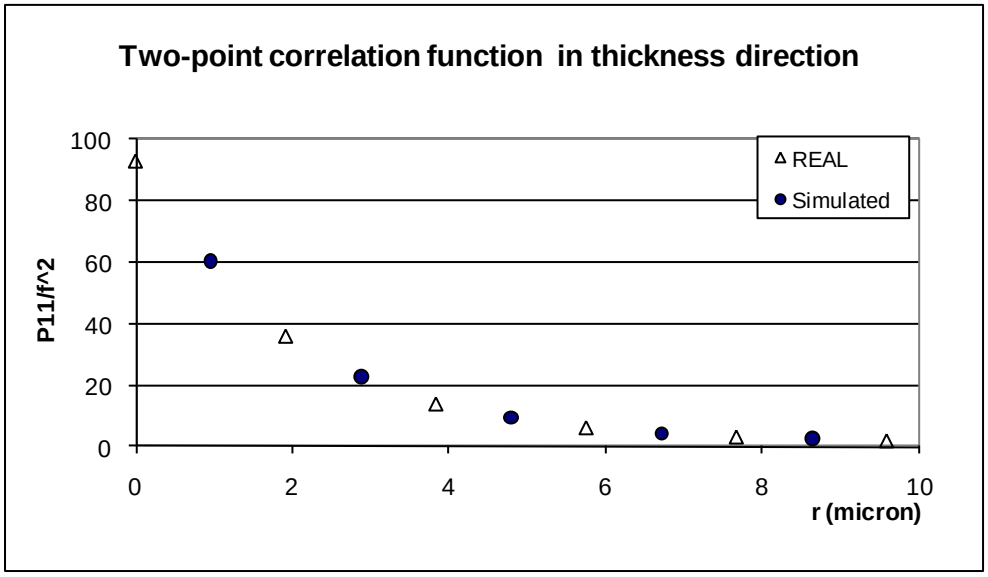


(i)

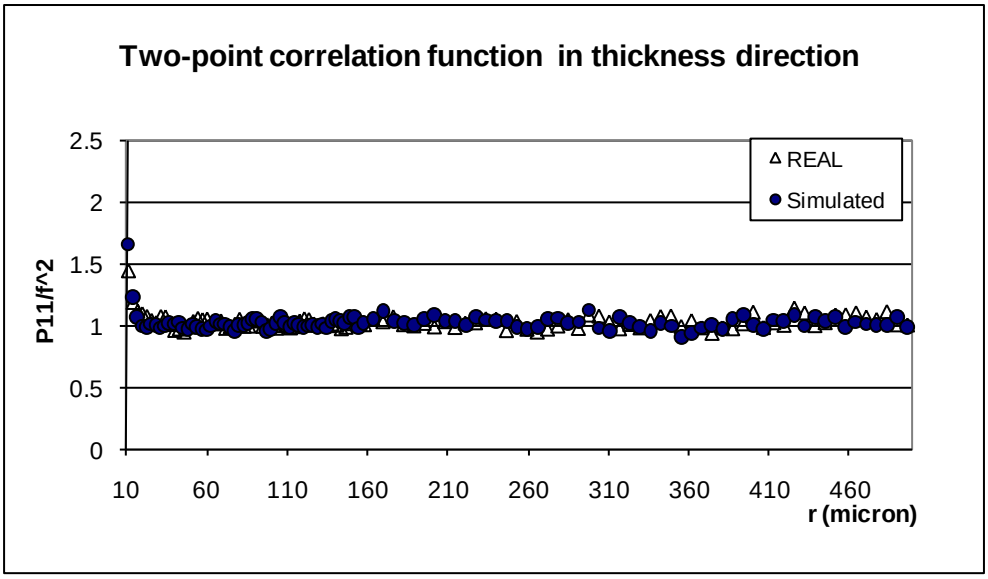


(ii)

Figure 3.17: Comparison of normalized two-point correlation functions of the constituent particles in the T-S plane along the transverse direction in real and simulated microstructures. (i) Short range data set, (ii) long range data set.

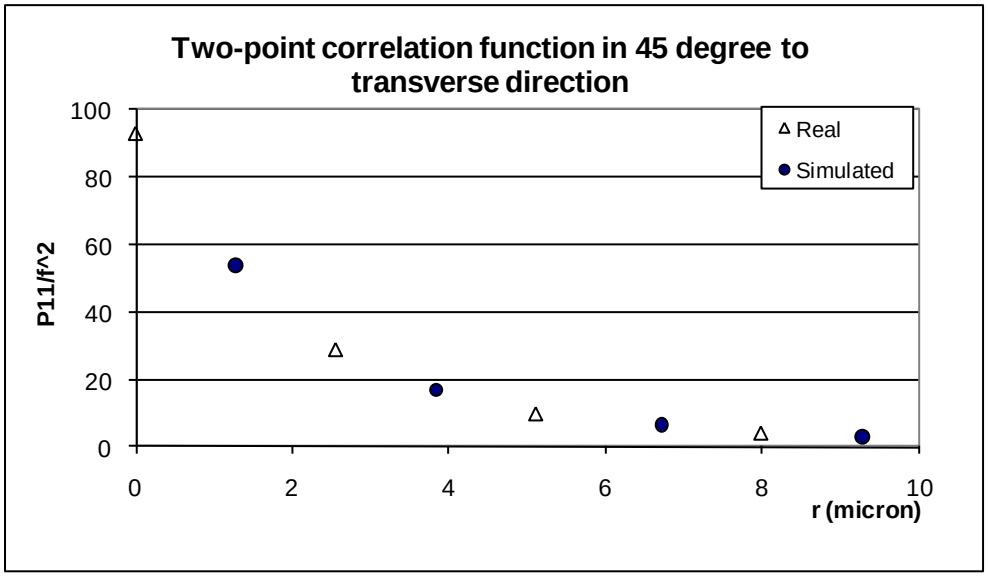


(i)

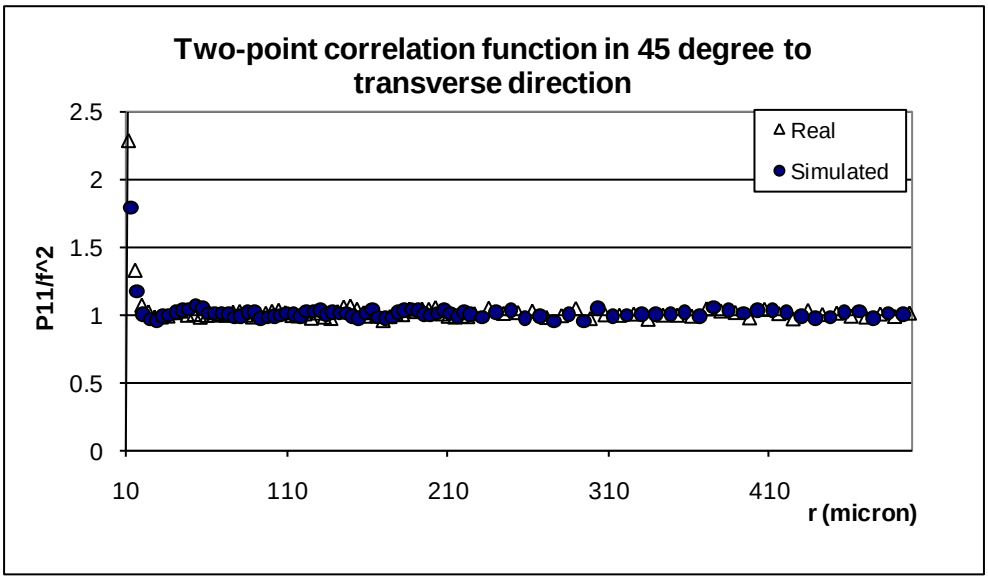


(ii)

Figure 3.18: Comparison of normalized two-point correlation functions of the constituent particles in the T-S plane along the thickness direction in real and simulated microstructures. (i) Short range data set, (ii) long range data set.



(i)



(ii)

Figure 3.19: Comparison of normalized two-point correlation functions of the constituent particles in the T-S plane along 45 degree to transverse direction in real and simulated microstructures. (i) Short range data set, (ii) long range data set.

These simulated microstructures are realistic because (i) they incorporate realistic complex particles shapes/morphologies as those present in the corresponding real microstructure, (ii) they account for realistic spatial clustering and anisotropy of the constituent particles, as represented by two-point correlation functions, similar to those in the corresponding real microstructure, (iii) they incorporate the same volume fraction and size distribution of the constituent particles as that in the corresponding real microstructure, (iv) they are sufficiently large so that the short-range (0 to 10 μm), intermediate-range (10 to 50 μm), and long-range (50 to 500 μm) spatial patterns and other microstructural details are represented at high resolution, and (v) they are generated by matching the simulated two-point correlation functions with the corresponding experimental data to ensure that spatial patterns along different directions are correctly represented. The simulation procedure is sufficiently flexible so that any specified extent of overlap can be permitted between the constituent particles, and if needed, the particles can be rotated to any specified extent to simulate any desired morphological anisotropy of the constituent particles.

It is important to recognize that in the present microstructure, the normalized two-point correlation function (particularly, along directions closer the rolling direction) does not reach the saturation value of 1.0 up to distance of 250 μm or so. Therefore, the length scales of these spatial patterns are quite large. On the other hand, the complex constituent particle morphologies represent intricate short-range microstructural details that also affect the mechanical properties of these microstructures. As all of these microstructural aspects are incorporated in these simulated microstructures, they are useful to serve as representative microstructural windows for the finite elements (FE)

based computational simulations of mechanical response of the corresponding microstructures.

The geometry of the simulated microstructures (and therefore, of corresponding real microstructures) is represented by the following simulation parameters used for the computer simulations.

- Volume fraction of constituent particles
- Size and shape distribution of the constituent particles
- Sizes, shapes, orientations and number densities of three types of bands (particle rich regions)
- Clustering intensity represented by the ratio of the number density of the constituent particles in the particle rich regions and the overall average global number density of constituent particles.

The values of these parameters for the present simulation model are given in Table 3.1. Note the aspect ratios of some type of cluster bands are large enough that for practical purpose they can be considered as lineal stringers. And also note that both L-S and L-T planes share the particle cluster band's maximum dimension, thus both simulated planes share the same major axis length parameters. The cluster parameters such as lengths of major and minor axis can be useful for simulating 3D microstructures. Figure 3.20 shows one way of simulating these particle rich regions (clusters) in the 3D space using parameters from Table 3.1.

Table 3.1: Values of simulation parameters used to generate simulated microstructures having specified correlation functions

	L-S Plane			**L-T Plane**			**S-T Plane**	
Number density of particles	650 mm^{-2}			527 mm^{-2}			815 mm^{-2}	
Volume fraction of particles	1.1%			1.1%			1.1%	
Elliptical Cluster bands (particle rich regions)	Type 1	Type 2	Type 3	Type 1	Type 2	Type 3	Type 1	Type 2
Length of major axis (μm)	301	125	55	301	125	55	70	38
Length of minor axis (μm)	2.6	2.6	2.6	8.6	38	3.2	2.0	2.0
Number density (mm^{-2})	21	13	13	6.0	7.0	25	6.0	19
Clustering intensity	20	49	70	18	11	70	35	106
Orientation of clusters* (degree)	0	0	0	-5 to 5	-8 to 8	-8 to 8	-5 to 5	-5 to 5

* In L-S and L-T plane, the orientation of cluster is the angle between major axis of cluster and rolling direction. In S-T, it's the angle between major axis of cluster and transverse direction.

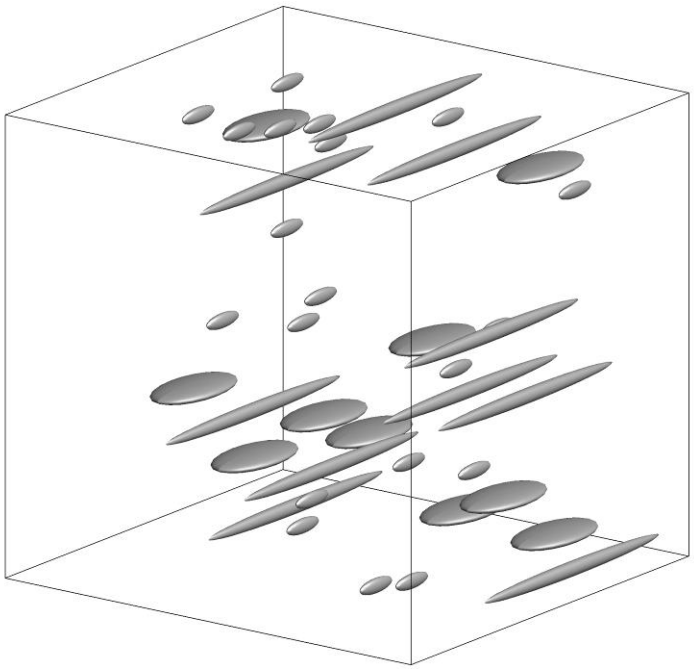


Figure 3.20: Microstructure model for constituent particle rich clusters in the 3D microstructure. Note that these are shapes of modeled particle rich regions and NOT the individual constituent particles.

3.4 Computer Simulated Virtual Microstructures

Depending on the simulation algorithm used, a simulated microstructure that closely matches the relevant attributes (feature size/shape distributions, volume fractions, spatial arrangements, anisotropy, etc.) of the corresponding *real* microstructure can capture the detailed relevant geometry of the real microstructure in terms of the values of simulation parameters used for the microstructural simulation. These microstructural simulation parameters can be then used to generate an atlas of rational "virtual" microstructures having a range of different constituent particle volume fractions and average particle sizes. Such virtual microstructures can be implemented in the computational models for materials properties and performance to identify the set of microstructures that are predicted to have a desired combination of material properties.

One can simulate an atlas of virtual microstructures of constituent particles that have different volume fractions and average particle sizes but the same spatial clustering and anisotropy. Such simulations mimic the changes in the corresponding real microstructures due to changes in percentages of impurity elements such as Fe and Si and the changes in the processing conditions**.** Figures 3.21 to 3.24 depict such simulated microstructures for two different volume fractions and two different average sizes of the constituent particles. Note that these virtual microstructures have the same realistic constituent morphologies, and represent realistic short-range (0 to 10 μm), intermediate-range (10 to 50 μm), and long-range (50 to 500 μm) spatial patterns and other microstructural details at high resolution. Such virtual microstructural windows can be used in the FE-based parametric studies to analyze the effects of constituent particle volume fraction, average size, etc on the micro-mechanical response, damage initiation,

and stress-strain curves of the virtual alloys. The resulting data can provide useful information for materials by design, and the methodology can reduce the number of experiments (and therefore, time and resources required) for developing new generation of wrought Al-alloys and for optimizing the properties of the existing alloys.

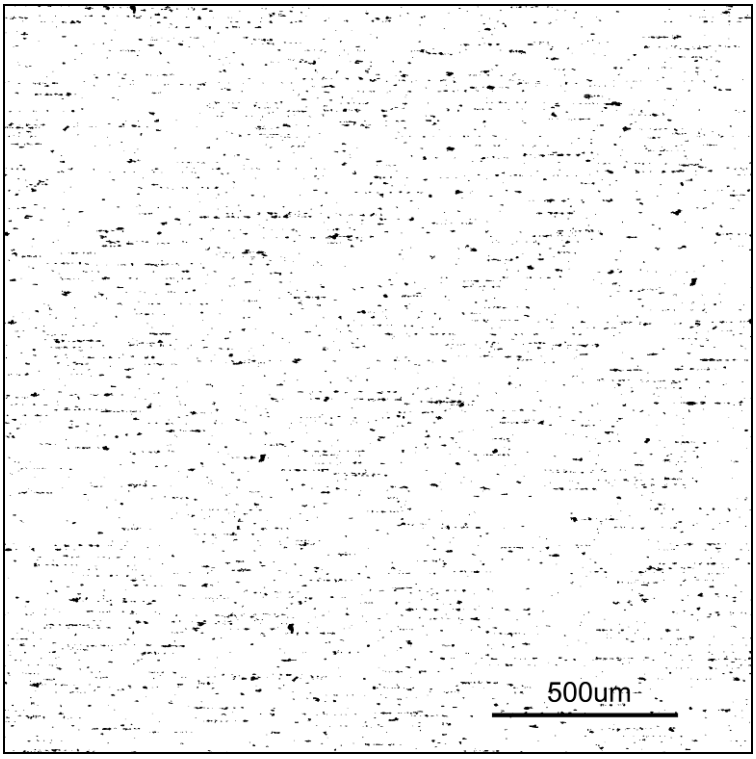


Figure 3.21: Simulated microstructure with higher volume fraction (2%) of the constituent particles

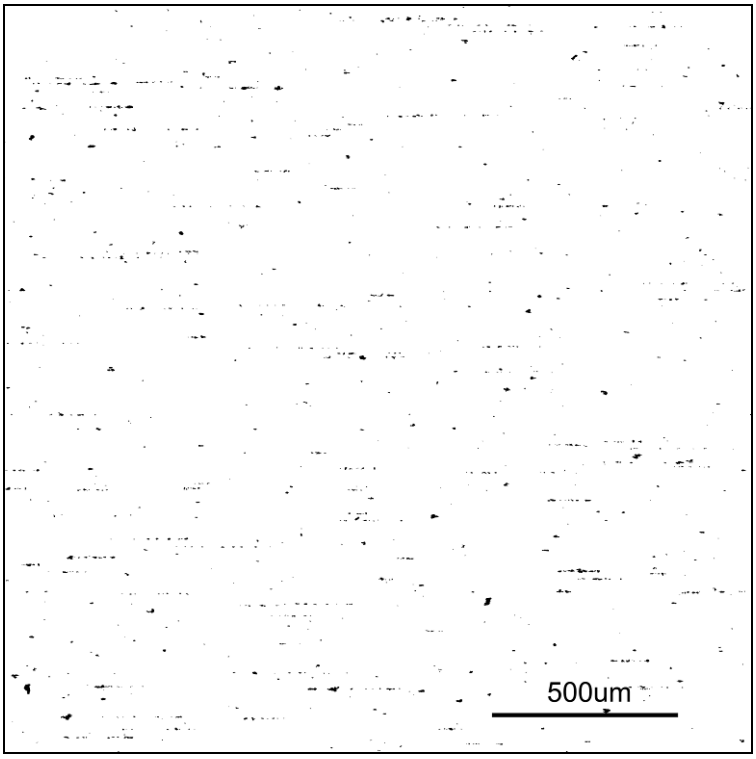


Figure 3.22: Simulated microstructure with lower volume fraction (0.5%) of the constituent particles

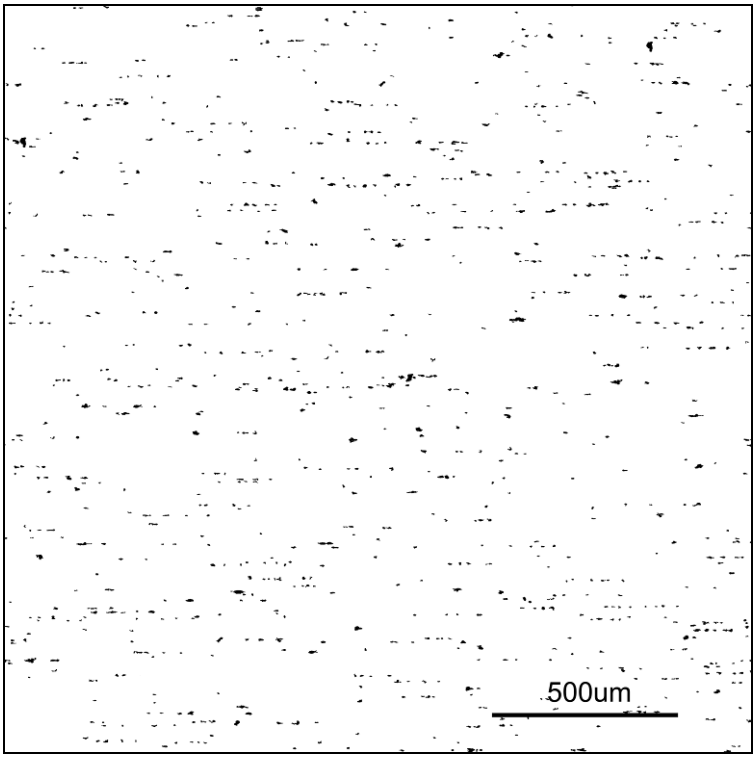


Figure 3.23: Simulated microstructure with larger average sizes (average feretmax = 15 μm) of the constituent particles

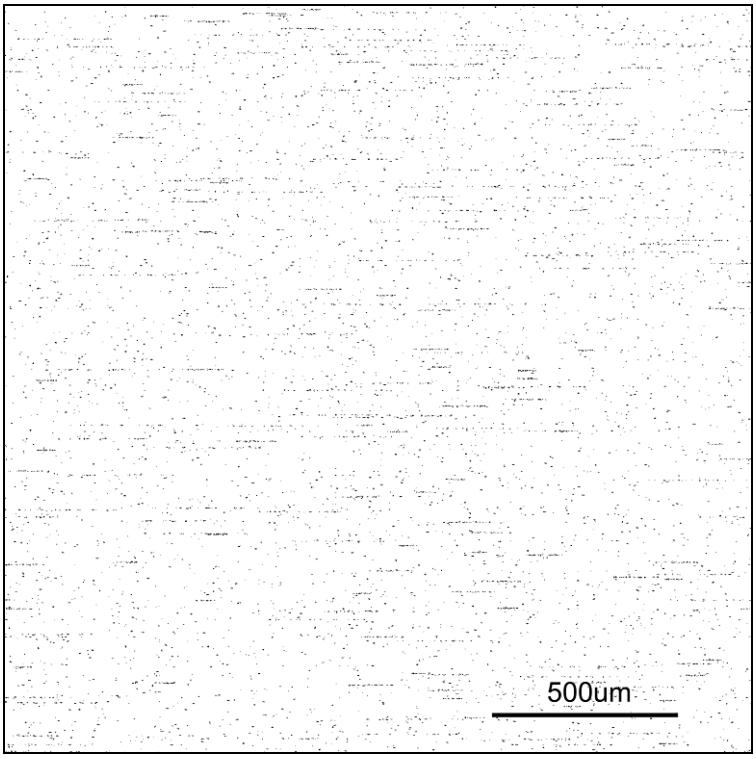


Figure 3.24: Simulated microstructure with smaller average sizes (average feretmax = 5 μm) of the constituent particles

3.5 Summary

A novel and efficient methodology is applied for computer simulations of realistic 2D microstructures of Fe-rich coarse constituent particles in a hot-rolled 70775 Al-alloy plate. The simulations incorporate realistic complex morphologies/shapes, spatial patterns, and size distributions of the particles. The methodology permits simulations of sufficiently large microstructural windows that incorporate short-range (on the order of particle/feature size) as well as long-range (hundred times the particle/feature size) microstructural heterogeneities and spatial patterns at high resolution (~0.2 μm pixel

size). The methodology also enables simulations of rational virtual microstructures of the alloys that have not been processed.

The simulated microstructures presented in this chapter represent the corresponding 2D microstructures observed in the L-S, L-T, and S-T metallographic planes of the alloy plates. It is important to recognize that a set of orientation dependent two-point correlation functions pertains to 3D microstructure, although estimated from the measurements on 2D sections, which has been proved using stereological arguments [68]. Therefore, if the 3D particle morphologies of the constituent particles are available, similar procedure can be used to simulate 3D microstructures by matching the same two-point correlation function data set. The methodology and applications for computer simulation of realistic 3D microstructures are presented in the next chapter.

CHAPTER 4

REALISTIC 3D MICROSTRUCTURE SIMULATION

4.1 Introduction

The central theme of the present research is development of methodology for computer simulations of realistic complex 3D microstructures that (1) incorporate realistic complex 3D particle/feature shapes, (2) allow controlled non-uniformities/clustering in spatial distributions of features, (3) permit partial anisotropic morphological orientations of microstructural features, (4) closely match *experimentally measured* attributes (spatial correlation functions, orientation distributions, size and shape distributions, volume fraction, etc.) of the corresponding real microstructures, and (5) efficiently generate sufficiently large segments of microstructure that contain short-range (on the order of particle/feature size), intermediate-range (five to ten times particle/feature size), and long-range (few hundred times the particle/feature size) microstructural heterogeneities and spatial patterns. Computer simulations of such complex two-dimensional microstructures of the constituent particles in a wrought Al-alloy were presented in the last chapter to illustrate the basic approach and to demonstrate the feasibility of development of such computer simulation methodology for three-dimensional microstructures. The important steps involved in the development of the methodology and its applications to 3D material microstructures are as follows.

- Collection of large volume high-resolution 3D digital microstructure data sets for microstructures of interest

- Mathematical representation of the reconstructed 3D microstructures using two- , three-, and four-point correlation functions
- Extraction of large number (~ few thousand) of 3D particle images from reconstructed microstructures
- Computer simulations of microstructures using real 3D particle images
- Variations of the simulation parameters of the simulated microstructures to match mathematical representations of the corresponding real microstructures.
- Generation of an atlas of virtual 3D microstructures of the materials of interest.

The methodology is developed through its applications to the 3D microstructures of discontinuously reinforced aluminum alloy (DRA) composites and boron modified titanium alloys and composites, and it is presented in the following sections.

4.2 Computer Simulations of Realistic 3D Microstructures of DRA Composites

4.2.1 Materials

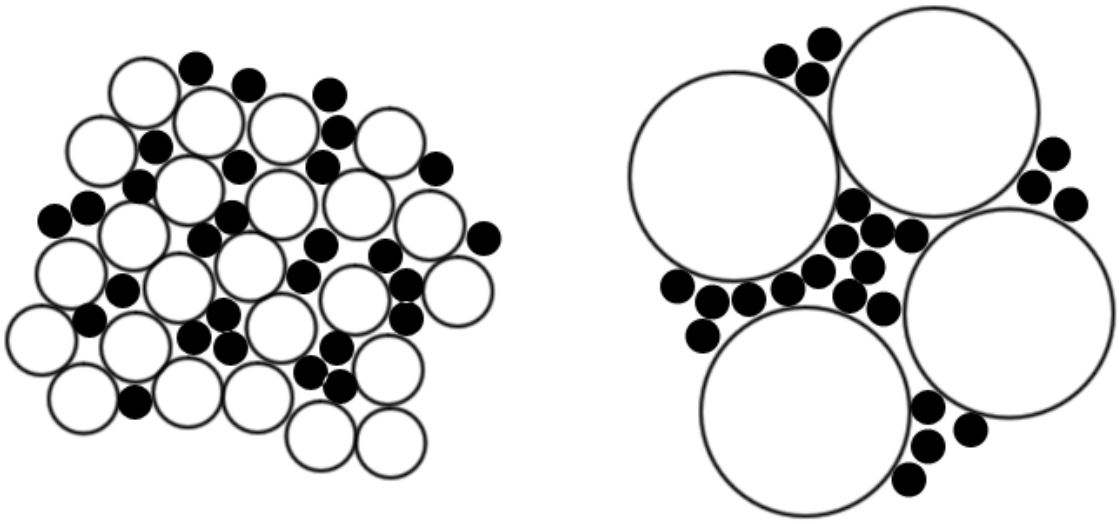
Discontinuously reinforced aluminum alloy (DRA) composites are a class of metal matrix composites (MMC) widely used in the aerospace, automotive, electronic packaging, and recreational product markets. The aluminum alloy matrix is reinforced with high strength constituents such as ceramic particles, whiskers, or short fibers. While various processing routes are available to manufacture DRA composites, a solid state processing technique of powder blending and consolidation (P/M processing) is one of the most common industrial methods used to produce composites having aluminum

matrix [101]. Powders of the alloy and the reinforcement are first blended and that is followed by cold compaction, canning, degassing, and high temperature consolidation steps such as hot isostatic pressing or extrusion.

The mechanical properties of the DRA composites depend on the microstructural attributes such as volume fraction, mean size, spatial clustering, and orientations/anisotropy of the reinforcement particles as well as on the constitutive behavior of the matrix that can be altered via heat treatment. The microstructures of the DRA composites are in turn governed by the process parameters of the P/M processing route including size/shape distributions of initial powders, compaction pressure, extrusion temperature, extrusion ratio, and relative amounts of the constituent powders. For example, an increase in the compaction pressure causes a reduction in the porosity in the composites [102]. An increase in the extrusion temperature increases the extent of anisotropy of the second phase particles and particle rich clusters, while reducing the overall porosity in the microstructure [103]. An increase in the extrusion ratio causes more uniform distribution of the reinforcement particles but also increases the extent of microstructural anisotropy [104].

One of the important processing parameters in the production of the DRA composites is the particle size ratio (PSR), which is defined as the ratio of the mean size of the matrix powder particles to the mean size of the reinforcement particles. It is known that PSR is an important factor that affects the spatial clustering of the reinforcement particles in the composites manufactured via powder metallurgy route [105]. Increasing the PSR (increasing the average size of the aluminum alloy matrix powder and keeping the SiC particle size the same) leads to a reduction in the combined surface area of the matrix

alloy particles and as this area becomes insufficient for a uniform arrangement of reinforcement particles, clusters of the reinforcement particles are formed in-between the larger matrix particles, as illustrated in Figure 4.1.



(a) Low PSR (b) High PSR

Figure 4.1: Comparison between low and high PSR values

In present research, the experiments have been performed on three different samples of extruded powder metallurgy processed discontinuously reinforced Aluminum alloy (DRA) composites containing SiC particles. All DRA composites contained the same F-600 grade SiC particles (median diameter, d_{50} = 13.4 μm). Figure 4.2 shows the size distribution of the SiC particles measured using a Microtrac X100 particle size analyzer. The median matrix particle size was varied in a controlled manner, by careful screening of the Al-6061 powder stock. The range of matrix particle sizes that were chosen (26.4 μm, 42.0 μm and 108.6 μm) produced DRA materials with increasing particle size ratios

(PSRs), which will be referred to as PSR = 2.0, PSR = 3.1 and PSR = 8.1, respectively. Extrusion was carried out on each sample at 450 ℃, with an extrusion ratio of 25:1 (round: round), followed by air-cooling. Further details of the material processing are given elsewhere [106]. Considering the processing conditions chosen in these experiments, it is reasonable to expect different degrees of spatial heterogeneity/clustering of the SiC particles due to the different PSRs, coupled with a moderate degree of anisotropy due to the moderate extrusion ratio employed.

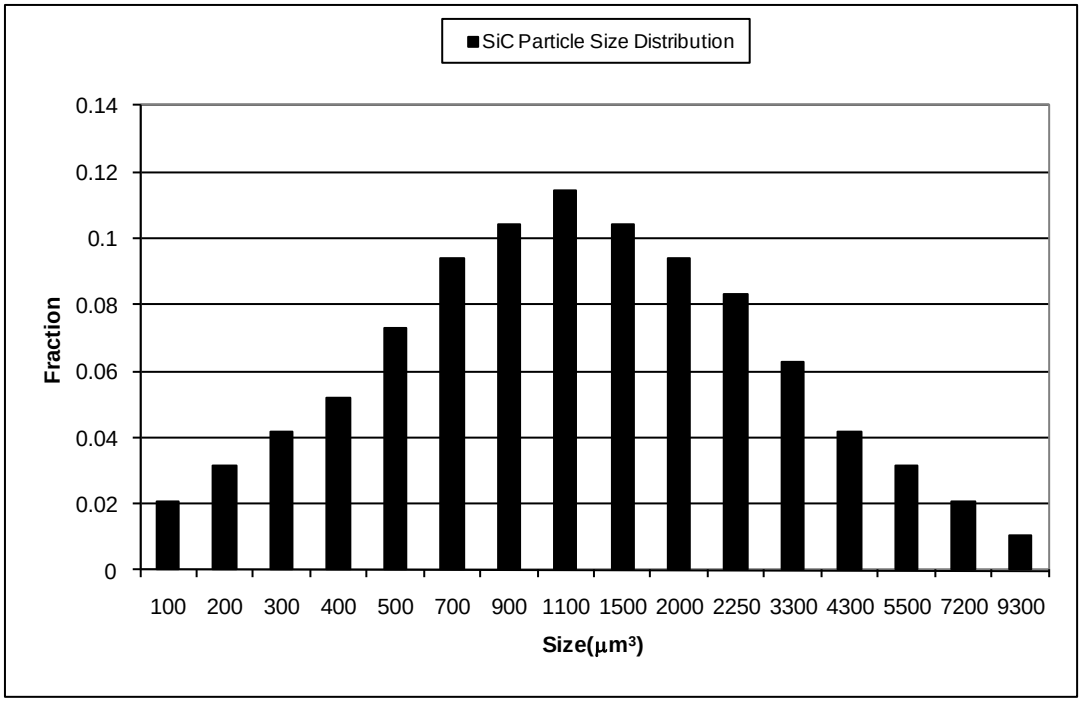
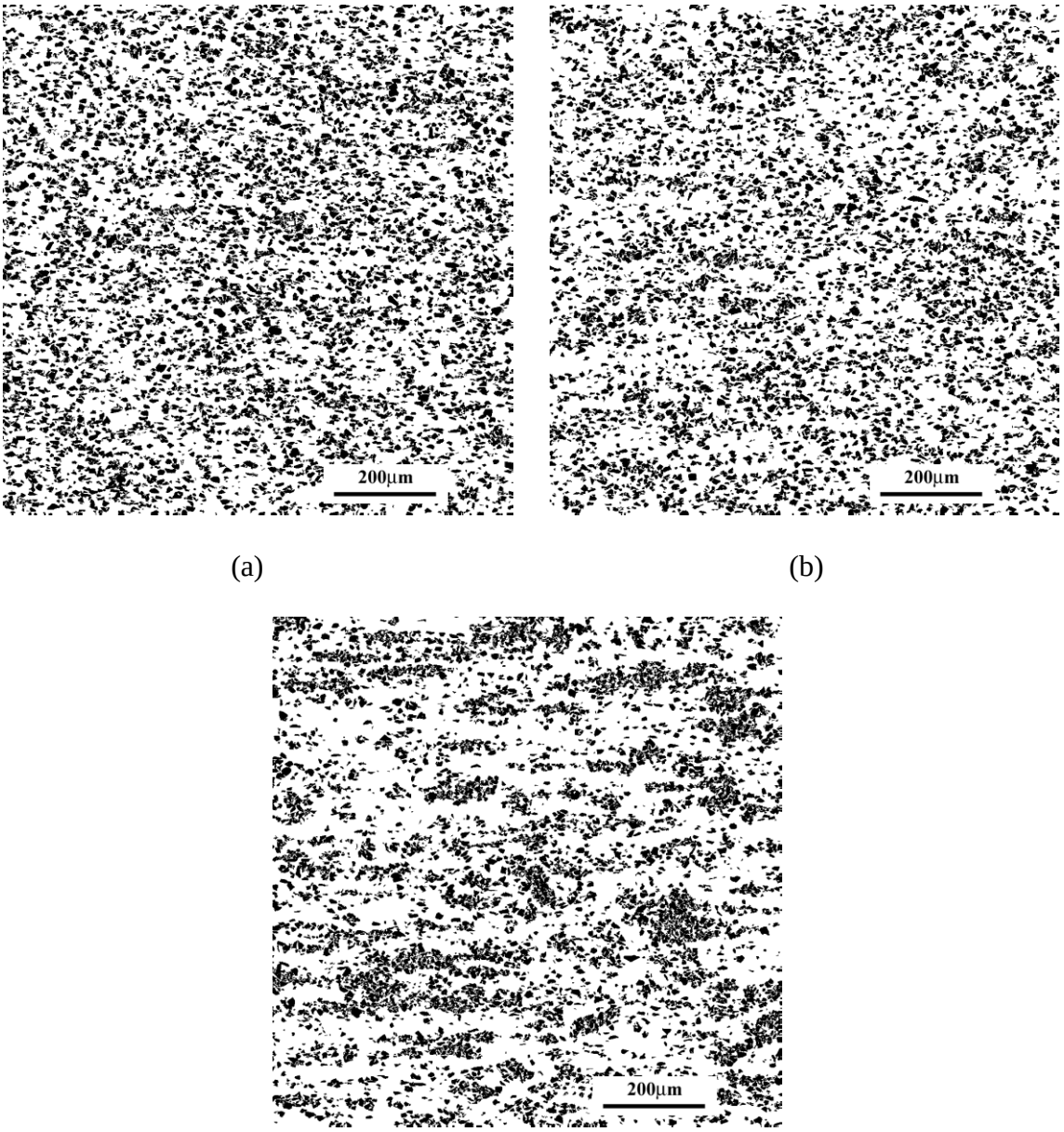


Figure 4.2: Size distribution of the SiC particles in the DRA composites

Figure 4.3 shows low-resolution micrographs of the longitudinal section of the DRA samples with PSR value of 2.0, 3.1, and 8.1. Note that the longitudinal section contains the extrusion axis of the sample. As expected, the sample with PSR 2.0 is

relatively homogeneous whereas long-range heterogeneity caused by clustering of SiC particles is observed with increasing PSR values.



(a) (b)

(c)

Figure 4.3: Low-resolution micrographs of the longitudinal section of the DRA samples with PSR value of (a) 2.0, (b) 3.1, and (c) 8.1

4.2.2 3D Microstructure Reconstruction

Depending on the material chemistry, processing, and microstructural length scales of interest, an opaque 3D microstructure can be rendered using numerous techniques, including classical serial sectioning [72], focused ion beam (FIB) tomography [73], atom probe tomography [74], magnetic resonance imaging (MRI), and x-ray tomography [75]. The montage serial sectioning technique [33, 107] is particularly suitable for microstructures that contain particles or features with significantly different length scales, and consequently, require reconstruction of a large volume segment of the 3D microstructure at a high resolution. In the present study, this technique has been applied to obtain 100 aligned montage serial sections for each DRA composite. The metallography and the 3D microstructure reconstruction of the DRA composites have been performed by Dr. Harpreet Singh [108], as a part of his doctoral thesis research. The detailed procedure of montage serial sectioning of these DRA composites has been reported elsewhere [33]

Figure 4.4 depicts a stack of 20 aligned montage serial sections for 8.1 PSR DRA composite microstructure (this figure has been digitally compressed for presentation). Each montage serial section has a size of 5000 × 5000 pixels at a resolution of 0.2 μm per pixel; see Figures 4.5 to 4.7 for PSR=2.0, 3.1 and 8.1 DRA composites, respectively. Figures 4.5b, 4.6b and 4.7b are high magnification views of the outlined region in Figures 4.5a, 4.6a and 4.7a, respectively. Note that the spatial clustering of SiC particles is not evident in high magnification images, which demonstrates that it is necessary to use montage serial sectioning to capture the microstructural heterogeneity and spatial patterns at all relevant length scales while at the same time to preserve information of complex

individual particle morphologies. The average distance between consecutive serial sections is approximately 1 μm. For computer simulations of realistic 3D microstructures, it is advantageous to have cubic voxels (3D analog of pixels) in which the resolution along the X, Y, and Z directions is the same. To achieve this, each experimental montage serial section image was resized (downsampled) to the size of 1000 × 1000 pixels at the resolution of 1 μm per pixel using Adobe Photoshop. This leads to a reconstructed 3D microstructure volume of 1000 × 1000 × 100 voxels with voxel size of 1 × 1 × 1 μm. Nevertheless, the resulting microstructure volumes are large enough to be useful for characterization and visualization of microstructure at small as well as large length scales.

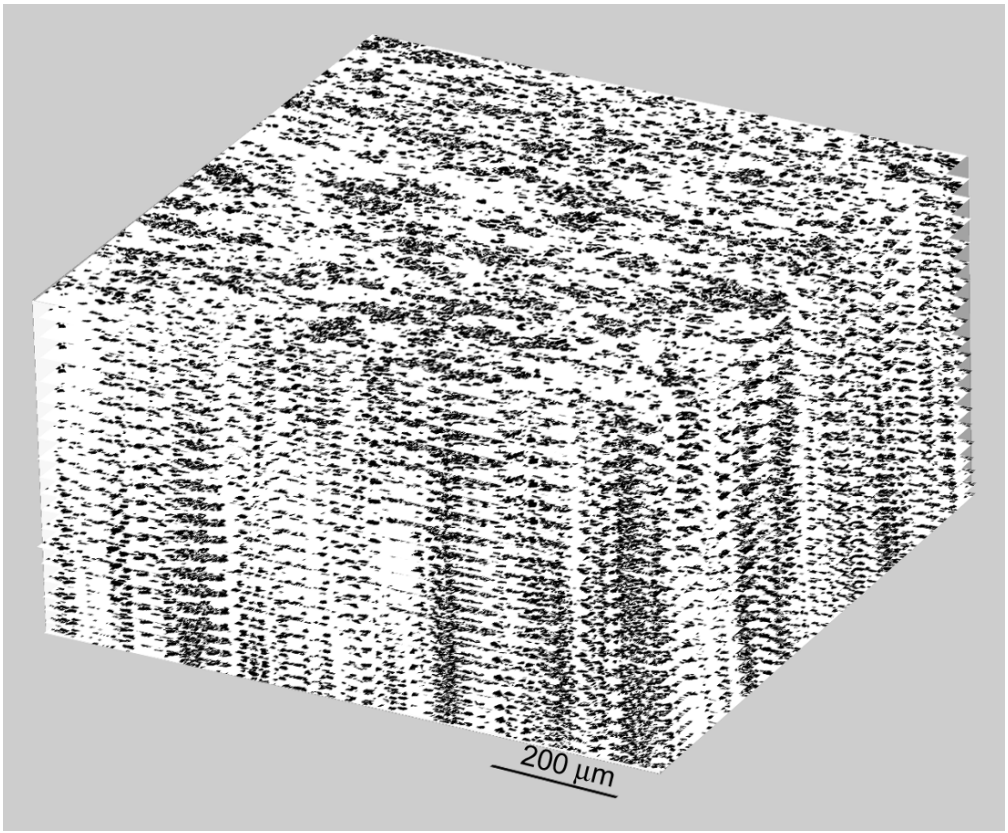


Figure 4.4: Stack of serial sections of 8.1 PSR DRA composite microstructure

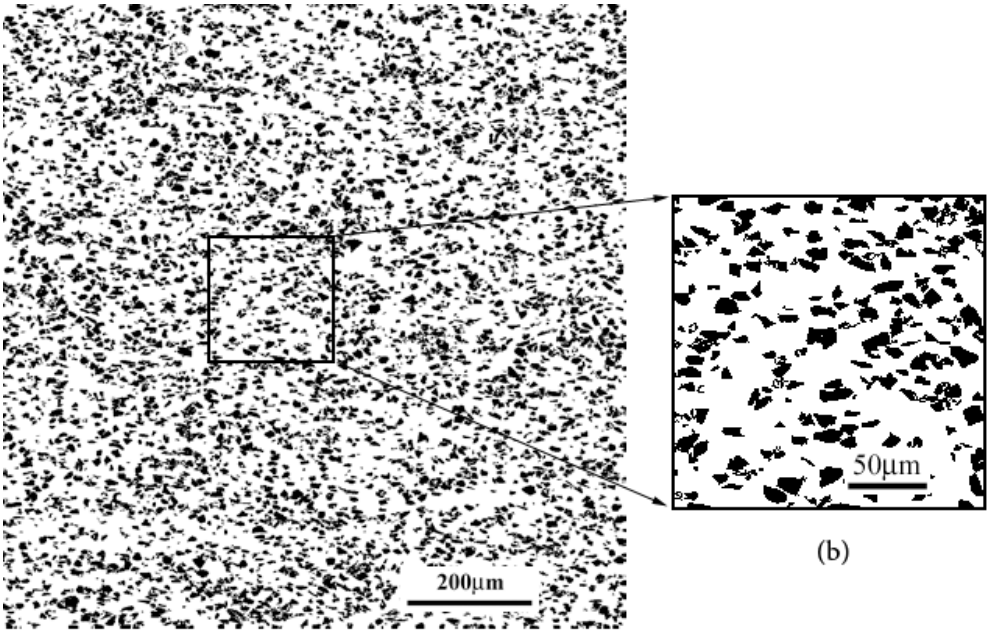


Figure 4.5: (a) Montage of 2.0 PSR DRA composite microstructure (b) Magnified view of the outlined region in (a)

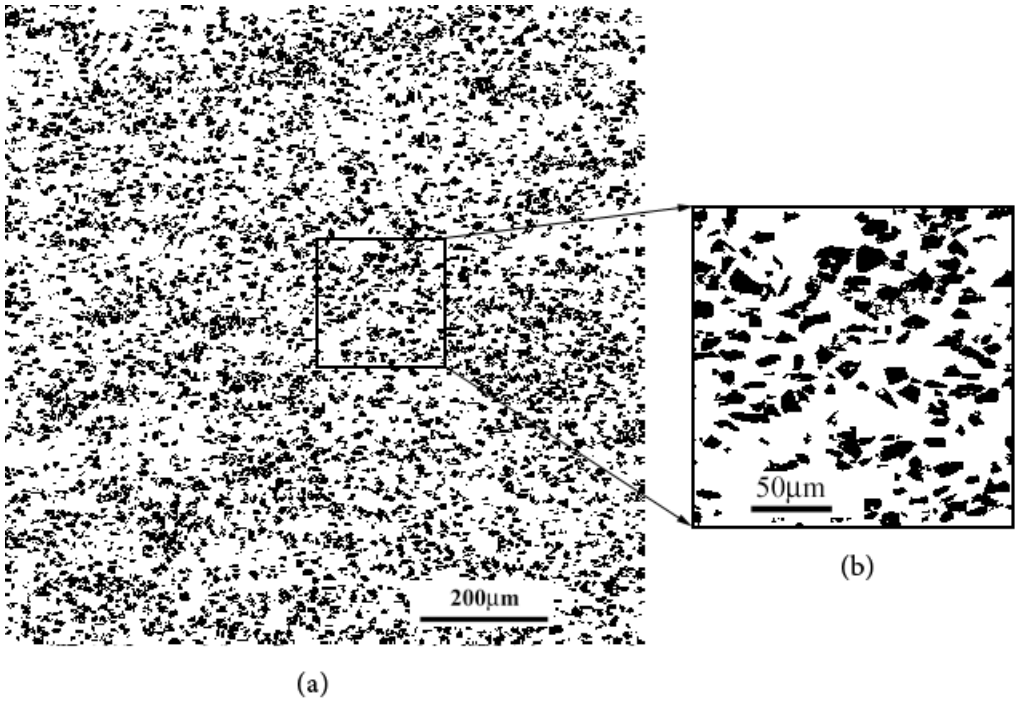


Figure 4.6: (a) Montage of 3.1 PSR DRA composite microstructure (b) Magnified view of the outlined region in (a)

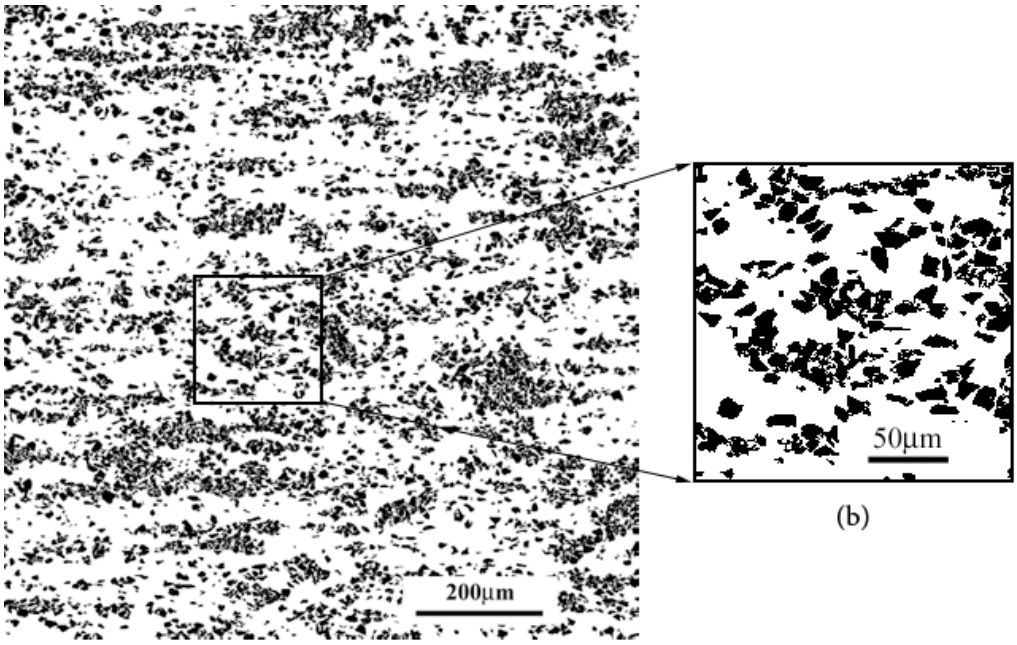


Figure 4.7: (a) Montage of 8.1 PSR DRA composite microstructure (b) Magnified view of the outlined region in (a)

A stack of aligned serial sections essentially constitutes a volume image data set similar to those encountered in X-ray computed tomography and MRI. Therefore, the same 3D microstructure visualization techniques are applicable. The 3D microstructural visualization can be achieved by surface rendering [79]. It involves rendering of the iso-surface of the region of interest (ROI) from the volume data, which leads to reduction in the size of the data set because only the surface data are retained. It is useful for examination of the 3D particle shapes and morphologies. Figures 4.8 to 4.10 show *small* segments of PSR 2.0, 3.1 and 8.1 DRA 3D microstructures reconstructed from the serial section images using surface rendering.

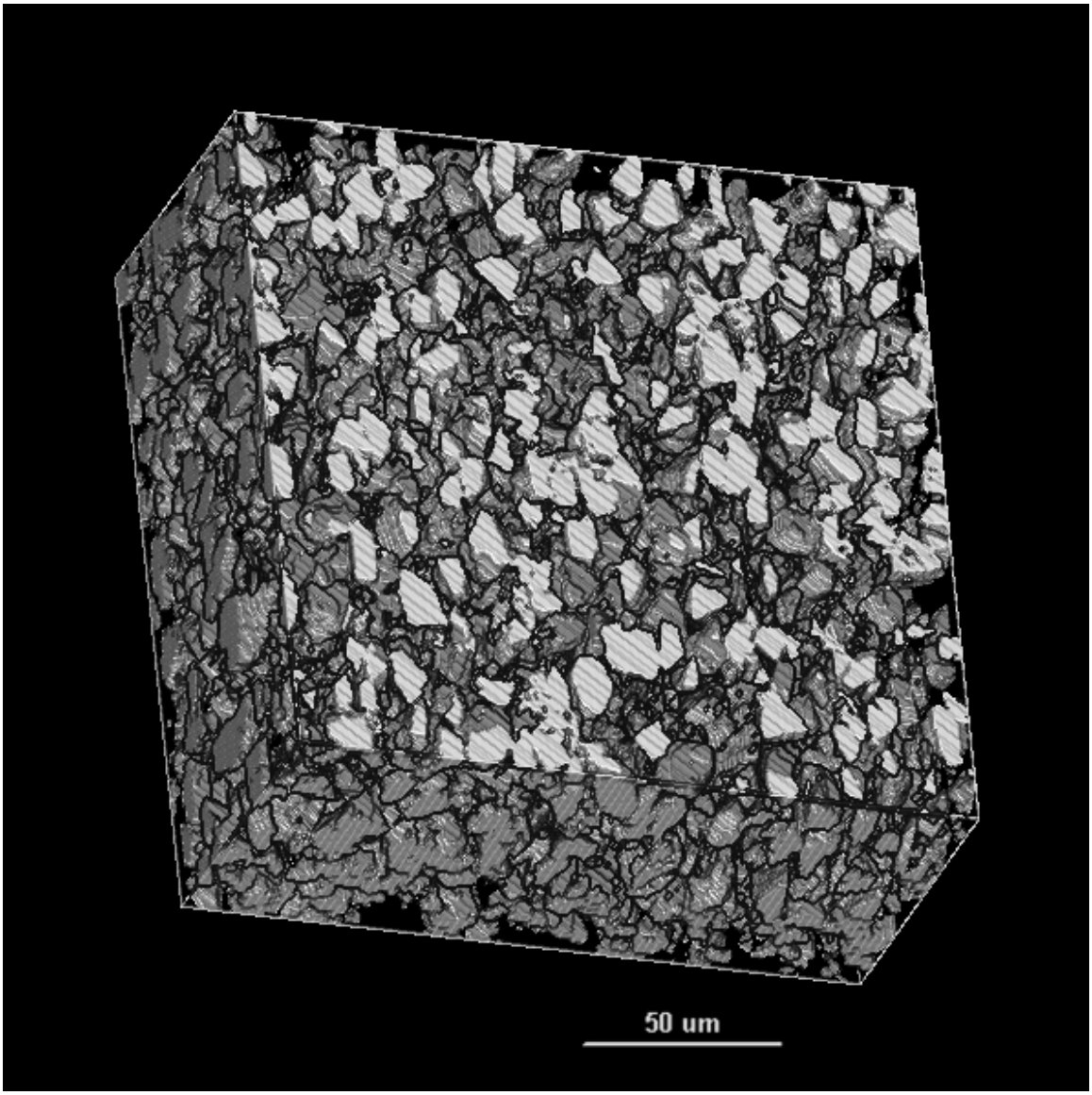


Figure 4.8: Small segment of the 3D microstructure of 2.0 PSR DRA composite reconstructed from the montage serial sections

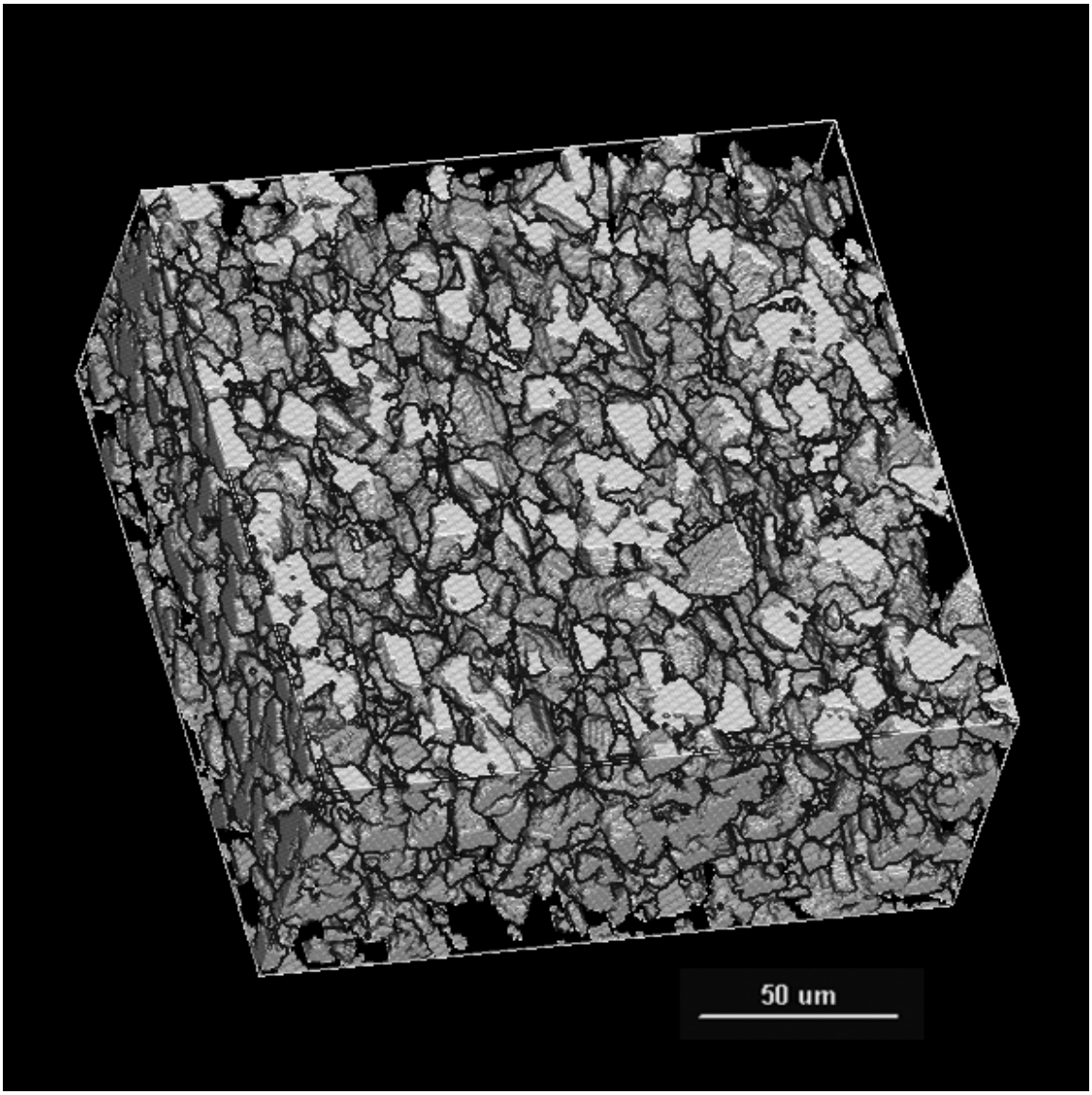


Figure 4.9: Small segment of the 3D microstructure of 3.1 PSR DRA composite reconstructed from the montage serial sections

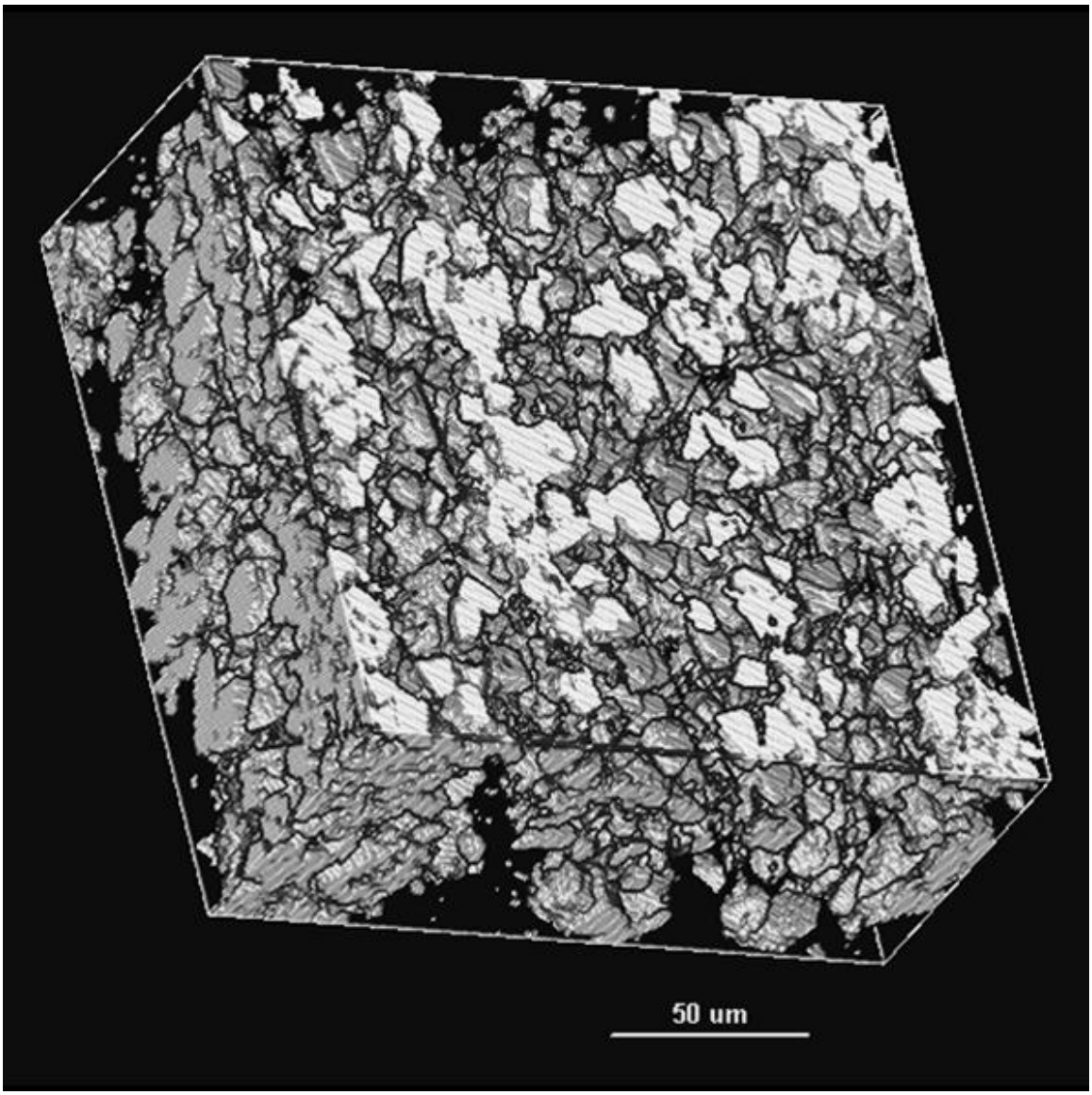


Figure 4.10: Small segment of the 3D microstructure of 8.1 PSR DRA composite reconstructed from the montage serial sections

4.2.3 Quantitative Microstructure Characterization and Representation

In order to be *realistic*, the simulated microstructure must closely match the relevant geometric attributes of the corresponding real microstructure, including spatial arrangements of the features, orientation distributions, size and shape distributions, volume fractions, etc. It is essential, therefore, to experimentally measure suitable

detailed statistical microstructural descriptors that are sensitive to these geometric aspects of the microstructure. As reported in Chapter 2, a microstructure can be statistically represented using numerous stochastic geometry based descriptors such as n-point correlation functions, lineal path probability distributions, nearest neighbor distributions, and radial distributions. In the present work, the two-point correlation function $P_{11}(r, \theta, \phi)$ is used for microstructure representation.

In general, $P_{11}(r, \theta, \phi)$ depends on the length of the probe line r as well as its angular orientation (θ, ϕ) in the 3D space. In the present microstructure, spatial clustering of the SiC particles is primarily due to the unequal sizes of the SiC and Al-alloy powder particles in the initial powder mix, whereas the directionality (anisotropy of the SiC particle-rich *clusters*) is attributable to the extrusion process. Due to the nature of the extrusion process, the anisotropy of the present microstructure (and therefore, of the estimated two-point correlation functions) is expected to be symmetric with respect to the extrusion axis. To take advantage of this symmetry, the extrusion direction is designated as the Z-axis, and therefore, for the two-point correlation function $P_{11}(r, \theta, \phi)$, θ is the angle between the probe line of length r and the Z-axis (extrusion direction), whereas ϕ represents the rotation of the line around this axis. As a result of the symmetry, all metallographic planes containing the extrusion axis exhibit statistically similar microstructures. Consequently, for the present microstructure, the two-point correlation function does not depend on ϕ. Its dependence on θ can be quantified by performing the measurements on a single metallographic plane containing the Z-axis, which is the extrusion direction. A robust digital image analysis and stereology based technique has been developed for the estimation of direction dependent two-point correlation functions

of any 3D microstructure from the measurements performed on vertical metallographic sections [68]. The technique permits precise and automatic estimation of the two-point correlation functions at distances ranging from 1 μm to 1000 μm (or more if needed) at a resolution on the order of 0.5 μm; the correlation functions can be estimated for all discrete line orientations in the vertical planes (planes containing Z-axis). This technique has been applied for estimation of the two-point correlation function of the DRA composite of interest in the present work.

Figure 4.11 shows the comparison of two-point correlation functions for the SiC phase (P_{11}) measured along the extrusion direction ($\theta = 0$) for the DRA microstructures with PSR values of 2.0, 3.1 and 8.1 on montage images of area 1000 μm × 1000 μm. The two-point correlation functions measured along the transverse direction ($\theta = \pi/2$) are shown in Figure 4.12. Note that the two-pint correlation data has been normalized by the square of volume fraction of SiC particles, f^2. By definition, the two-point correlation function approaches the square of volume fraction of the phase under consideration as r approaches infinity. This means that when the normalized two-point correlation function for SiC particles approaches 1.0 at length *r*, the particles at either ends of a line of length *r* are no longer spatially correlated to each other. In other words, the distance at which normalized two-point correlation function approaches 1.0 is an indicator of the characteristic length scale of the whole microstructure.

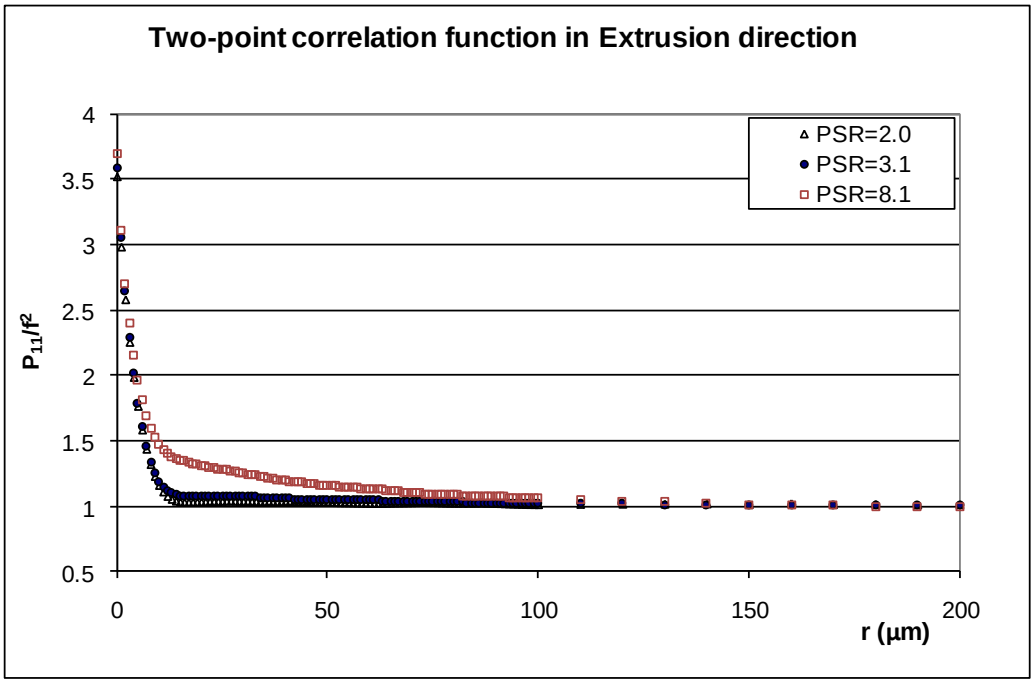


Figure 4.11: Two-point correlation functions for the DRA composites measured along the extrusion direction

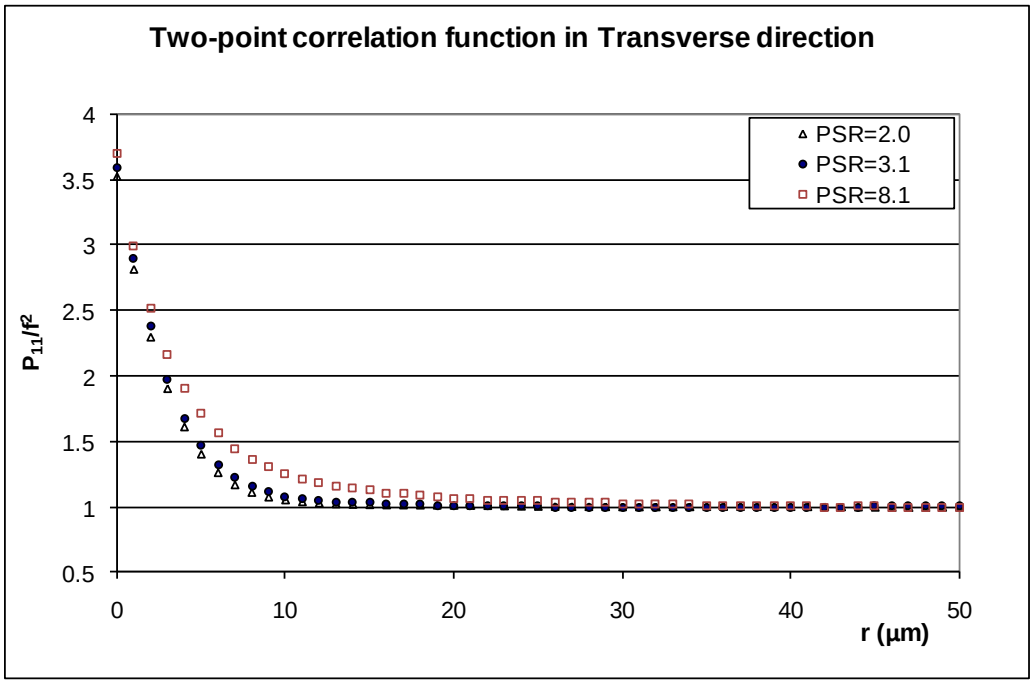


Figure 4.12: Two-point correlation functions for the DRA composites measured along the transverse direction

The two-point correlation function clearly brings out the differences in the spatial distribution of the SiC particles in the three samples. For both extrusion and transverse directions, the function approaches 1.0 at the shortest distance for the 2.0 PSR microstructure, followed by 3.1 PSR and 8.1 PSR microstructures. This is to be expected as the spatial distribution of SiC particles is more uniform in the 2.0 PSR microstructure and more clustered in the 8.1 PSR microstructure. For all the three microstructures, the two-point correlation function approaches 1.0 at a shorter distance along the transverse direction compared to the extrusion direction, which can be attributed to the anisotropy introduced by the extrusion process. These experimental data on the two-point correlation functions are used in the present work to compare the correlation functions of the real and corresponding simulated microstructures in order to ensure that the simulated microstructures are statistically representative of the corresponding real microstructures. The methodology for realistic simulations of these microstructural windows is presented in the next section.

4.2.4 Computer Simulations of 3D Microstructures

In the present work, a combination of digital image processing and microstructure simulation algorithms is used for the simulations of 3D microstructures with realistic complex particle shapes/morphologies. Conceptually, the simulation procedure is as follows. Focus on a set of high magnification high-resolution 3D microstructural binary serial section images such as the one showed in Figure 4.8. Such a microstructural volume obviously contains the real 3D morphologies of SiC particles. Consider a thought experiment where a large number of SiC particles are simply “plucked” out from the microstructural 3D volume and stored in a box such that the set is representative of the

size, shape, and morphology distributions of the SiC particle population in the microstructure of the real material. Next, thoroughly "shake" the box containing the plucked out SiC particles, take out SiC particles at random, and place them (as per some specified spatial arrangement and number density) using RSA [25] and/or Metropolis [26] algorithms in a digitized simulation space where voxel size is the same as that in the microstructural volume from which the SiC particles are plucked out. The result is a simulated microstructure containing the same SiC particle morphologies as those in the corresponding real microstructural images but (depending on the simulation parameters and algorithm) a different spatial arrangement of the particles, different particle volume fraction, average size, number density, anisotropy, etc. The important steps of this methodology are described in detail as follows.

4.2.4.1 Capturing Real 3D Particle Morphologies

The set of (*X, Y, Z*) coordinates of all points (voxels) inside of a particle contains complete detailed information on that particle's morphology and geometry. Once such a set of points is available, the exact replica of that particle can be then reproduced at any desired location in the simulation space. In the present study, a C++ computer code using 3D connected components labeling algorithm based on label equivalences [109] has been developed to label and extract the voxels of SiC particles. The computer code is given in Appendix B.6. The code scans a 3D volume, voxel by voxel in the raster scanning order (incrementing x first, followed by y, and then z), and groups its voxels into particles (components) based on voxel connectivity (26-connectivity), i.e. all voxels in a particle (connected component) share similar voxel intensity values and are in some way connected with each other. Once all particles (groups) have been determined, each voxel

is labeled with a number according to the particle (component) it belongs to. In this way, the code generates the sets of coordinates/locations of the voxels that form each particle. Using this image analysis procedure, 500 SiC particles were extracted to represent the size and shape distribution of the SiC particle population observed experimentally. This SiC particle size distribution matches the data in Figure 4.2. The data set contains the positions of all voxels of each particle. The centroid voxel position of each SiC particle is then computed from the positions of the voxels on the boundary. Using simple coordinate translation (change of origin), the (*X*, *Y*, *Z*) positions of the voxels are changed so that each particle centroid is at a (0, 0, 0) position. Now, the placement of a particle at a location in the simulation space simply involves translation of the particle centroid from (0, 0, 0) to the voxel coordinates of the new location in the simulation space. The data set corresponding to each particle is then stored in the computer memory. Next, each of the N particles is assigned a distinctive number in the range of 1 to N to identify that particle; these identification numbers are assigned in a random manner, and they have no correlation to the size or shape of the particles. Figure 6 shows some 3D SiC particles extracted in this manner.

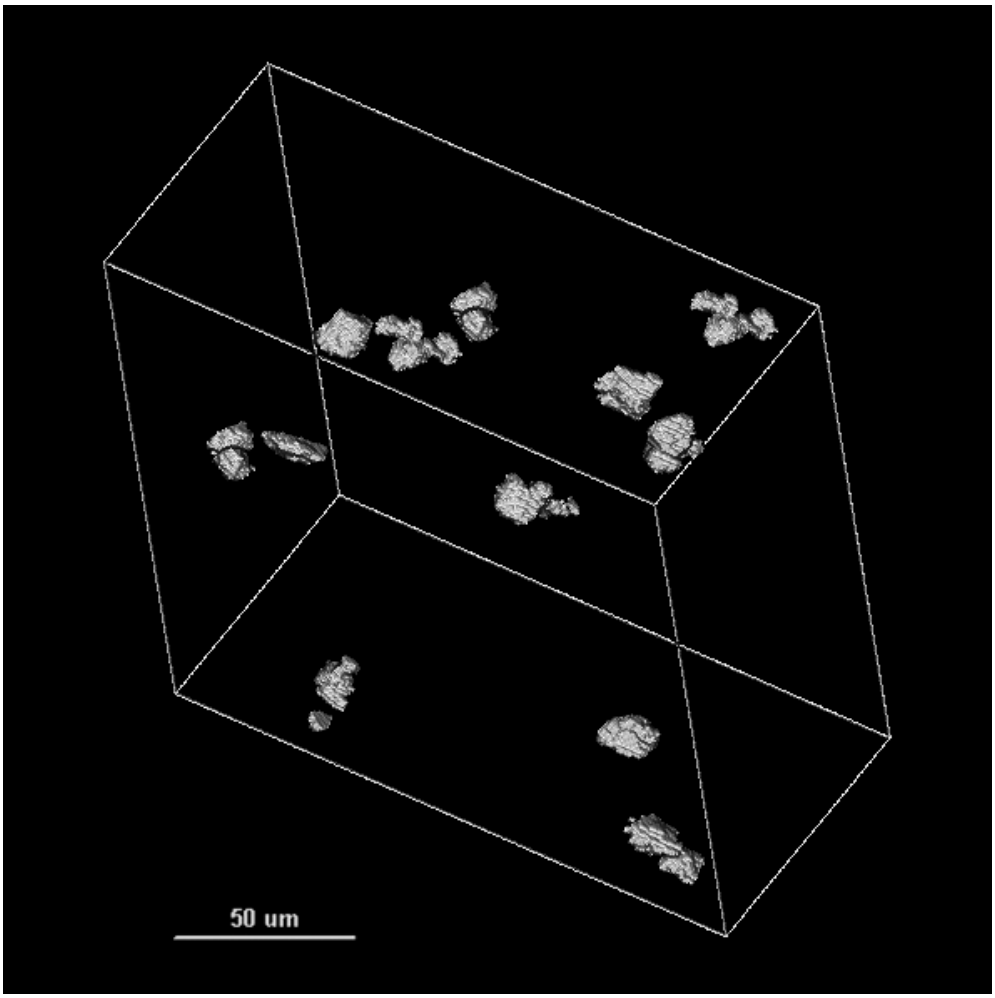


Figure 4.13: SiC Particles extracted by 3D component labeling

4.2.4.2 Simulation of Locations of Particle-rich and Particle-poor regions in 3D Simulation space

In the present composite, the spatial clustering of SiC particles is primarily due to large value of the particle size ratio (PSR), whereas the directionality observed in the majority of the SiC particle clusters is primarily due to the extrusion process. Due to the extrusion process, majority of the particle clusters are elongated to large extent (i.e. high aspect ratio). Nonetheless, as the deformation due to the extrusion process is not

necessarily uniform at all locations, some SiC particle clusters have low aspect ratio and are relatively less elongated. Therefore, two types of cluster regions (representing SiC particle rich regions) are first simulated, namely, the high aspect ratio and low aspect ratio regions. The number densities of the two types of regions, their size, and aspect ratios are important simulation parameters. These regions are simulated in shape of prolate spheroids where the polar diameters are aligned with DRA's extrusion direction, and placed in the simulation space using the well-known RSA algorithm [25, 84]. These ellipsoids representing the SiC particles rich clustered regions are not permitted to overlap and their centers are at uniform random locations in the 3D simulation space. The computer code is given in Appendix B.7.

4.2.4.3 Placement of SiC particles in the Simulation Space

Next, SiC particles are placed within each ellipsoidal cluster region with a specified number density $[N_V]_{cluster}$. Due to the high volume fraction (larger than the RSA 3D jamming limit [26]) within each cluster, the SiC particles are placed using the Metropolis algorithm (MA). At the beginning, all particles are placed in a 3D FCC lattice. And then, each particle is allowed to take a n-step (n>1000) random walk where the particles may overlap with other particles to a limited extent (as in Cherry-Pit Models [16]). The resulting microstructure has no memory of the initial lattice structure.

Next, SiC particles are simulated at uniform random locations with a different number density $[N_V]_{poor}$ at all locations of the simulation space not covered by the clustered regions. The intensity of the SiC particle clustering is then given by the parameter $[N_V]_{cluster}/[N_V]_{global}$, where $[N_V]_{global}$ is the global average number density of the SiC particles in the simulated microstructure. This process of placement of particles is

iterated until desired overall volume fraction and size/shape distribution of particles is achieved. The computer code is given in Appendix B.8. Note that in the present simulations, the particle orientations are kept the same as those in the corresponding real microstructure. However, the computer code permits particle rotations, if needed. This feature can be used to simulate a microstructure having specified morphological anisotropy of particles, which will be presented in detail in the subsequent section.

4.2.4.4 Use of correlation functions to compare real and simulated 3D microstructures

In the present work, experimentally measured two-point correlation functions $P_{11}(r, \theta, \phi)$ along the extrusion direction, the long transverse direction and the short transverse direction are used to represent the 3D microstructure of the real material [68, 91], and then the simulated microstructure is varied till the two-point correlation functions of the simulated microstructure $[P_{11}(r, \theta, \phi)]_{sim}$ closely match experimentally measured functions $[P_{11}(r, \theta, \phi)]_{exp}$ for the corresponding real microstructure. The simulated microstructure can be altered by changing the simulation parameters such as (1) number densities of the clusters, (2) size and shape of clusters, (3) intensity of clustering of SiC particles (i.e. $[N_V]_{cluster}/[N_V]_{global}$), (4) extend of overlap between SiC particles, etc.

At any given values of r, θ and ϕ, the absolute fractional error $E(r, \theta, \phi)$ can be computed as follows.

$$E(r,\theta,\phi) = \frac{|[P_{11}(r,\theta,\phi)]_{sim} - [P_{11}(r,\theta,\phi)]_{exp}|}{[P_{11}(r,\theta,\phi)]_{exp}} \qquad \text{Eq. 5}$$

Let $< E(\theta, \phi) >$ be the average value of $E(r, \theta, \phi)$ averaged over all values of r for given direction (θ, ϕ). In the present work, a simulated microstructure is considered to be representative of the corresponding real microstructure, if and only if, (1) $< E(\theta, \phi) >$ is less than or equal to 0.04 for each direction (θ, ϕ) of interest, and (2) For any give value of r, θ and ϕ, $E(r, \theta, \phi)$ is less than or equal to 0.06. Thus, the process involves numerous iterations of simulated microstructures with different combinations of the simulation parameters till the above two conditions are satisfied. In this way, the simulated microstructure has realistic particle morphologies as well as spatial patterns, size distribution, and volume fraction. In the present work experimentally measured two-point correlation function has been used for microstructures representation, but the procedure is equally applicable for simulations of microstructure having any specified two-point correlation function (or a higher order correlation function), size distribution, and volume fraction of particles.

4.2.5 Computer Simulations Results

The new realistic 3D microstructure simulation technique was applied to simulate the 3D microstructures of 2.0, 3.1 and 8.1 PSR DRAs containing spatially clustered SiC particles. An input volume of $200 \times 200 \times 100$ voxels of the real microstructure was used to extract the real particle morphologies. The simulation code was applied on these input particles to generate three simulated 3D microstructural volumes having a volume of $1000 \times 1000 \times 100$ voxels with voxel size of $1 \times 1 \times 1$ μm and each containing over seventy thousand particles. Figures 4.14 to 4.16 depict small segments of microstructural volume of PSR 2.0, 3.1 and 8.1 DRA simulated in this manner, which are comparable to

the real 3D microstructures shown in Figures 4.8 to 4.10, respectively. Each simulated volume can be divided into 100 simulated montage serial sections having a size of 1000 × 1000 pixels at a resolution of 1 μm per pixel (see Figures 4.17 to 4.19).

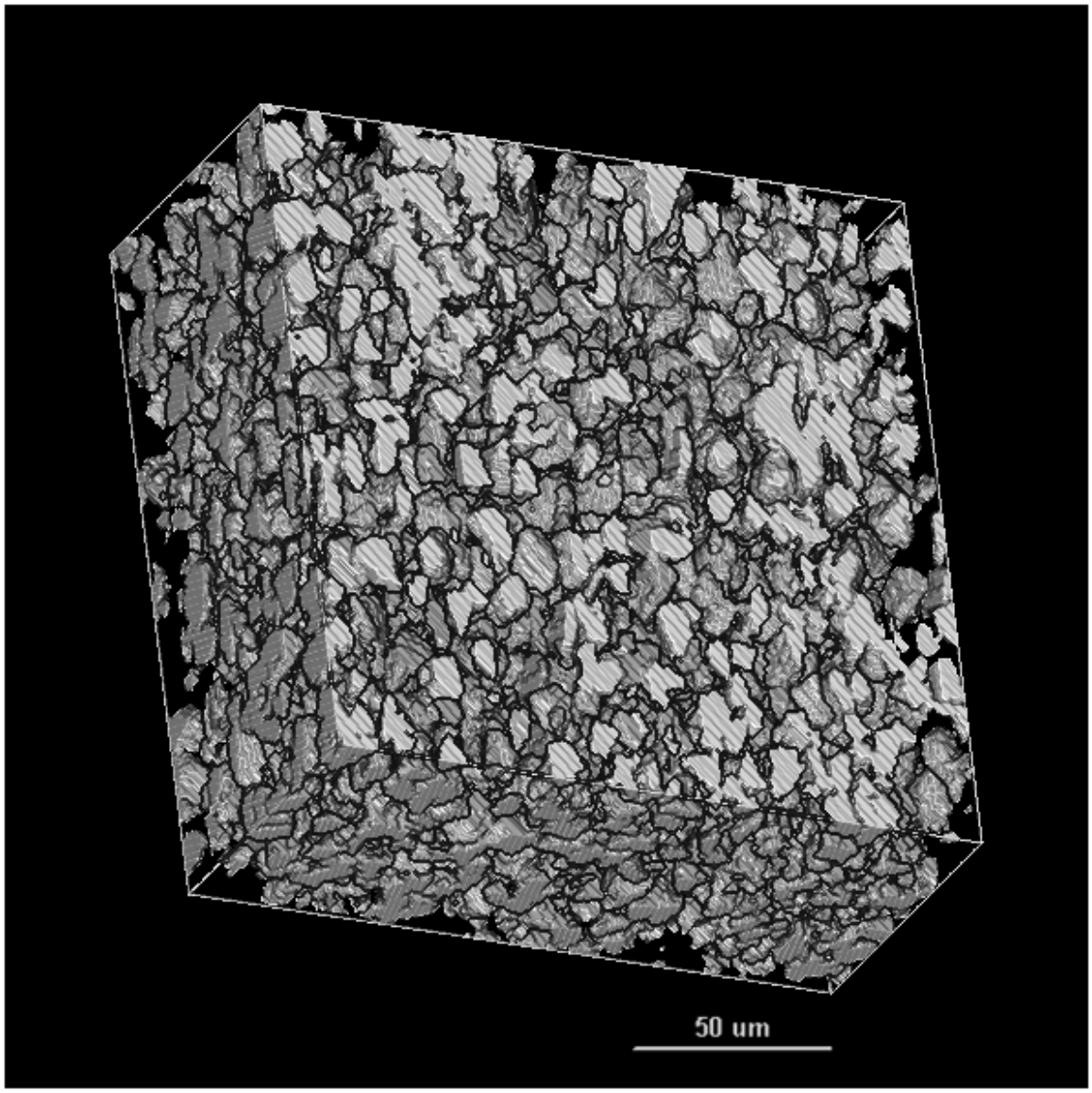


Figure 4.14: Small segment of simulated 3D microstructure of 2.0 PSR DRA composite

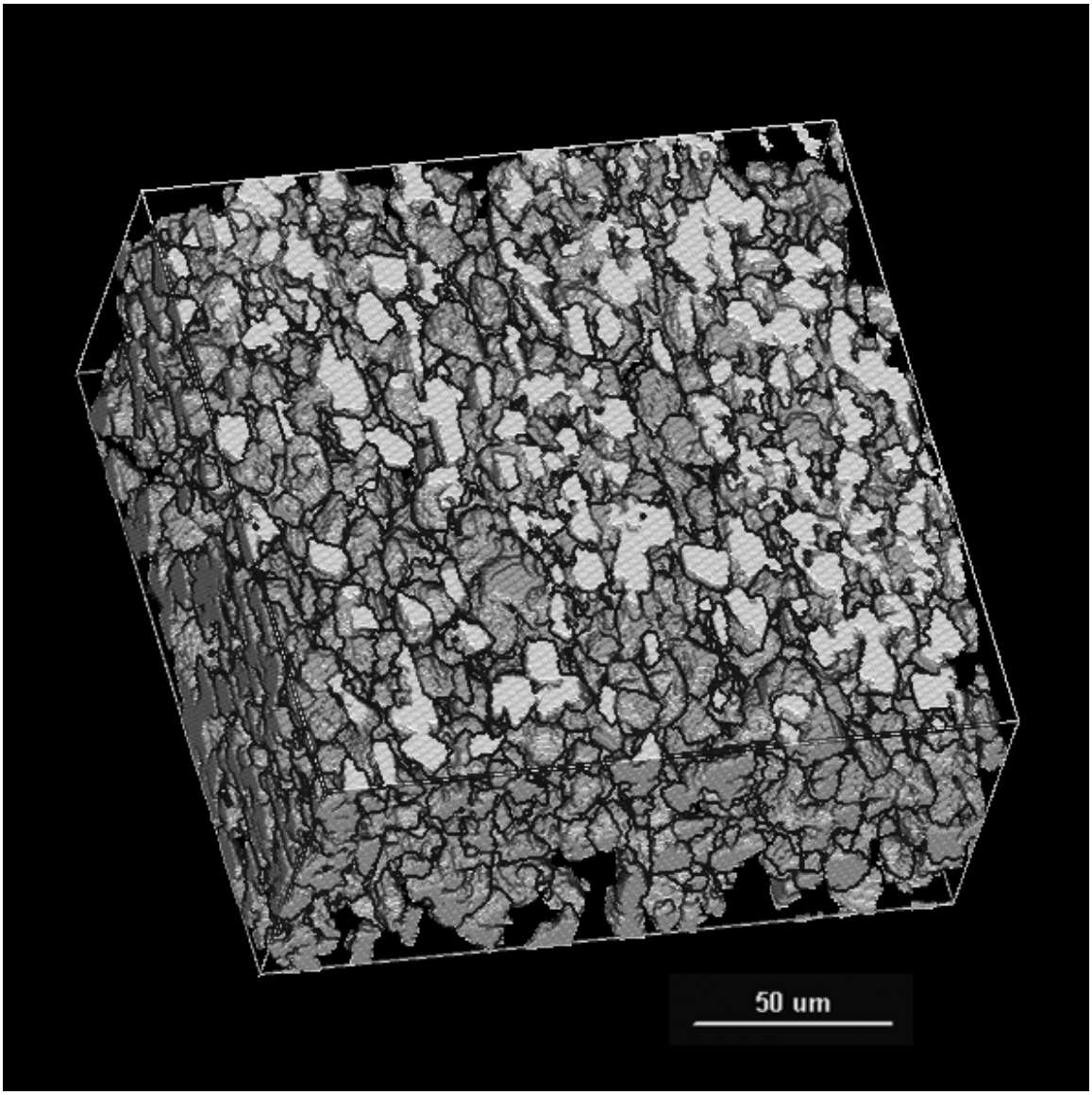


Figure 4.15: Small segment of simulated 3D microstructure of 3.1 PSR DRA composite

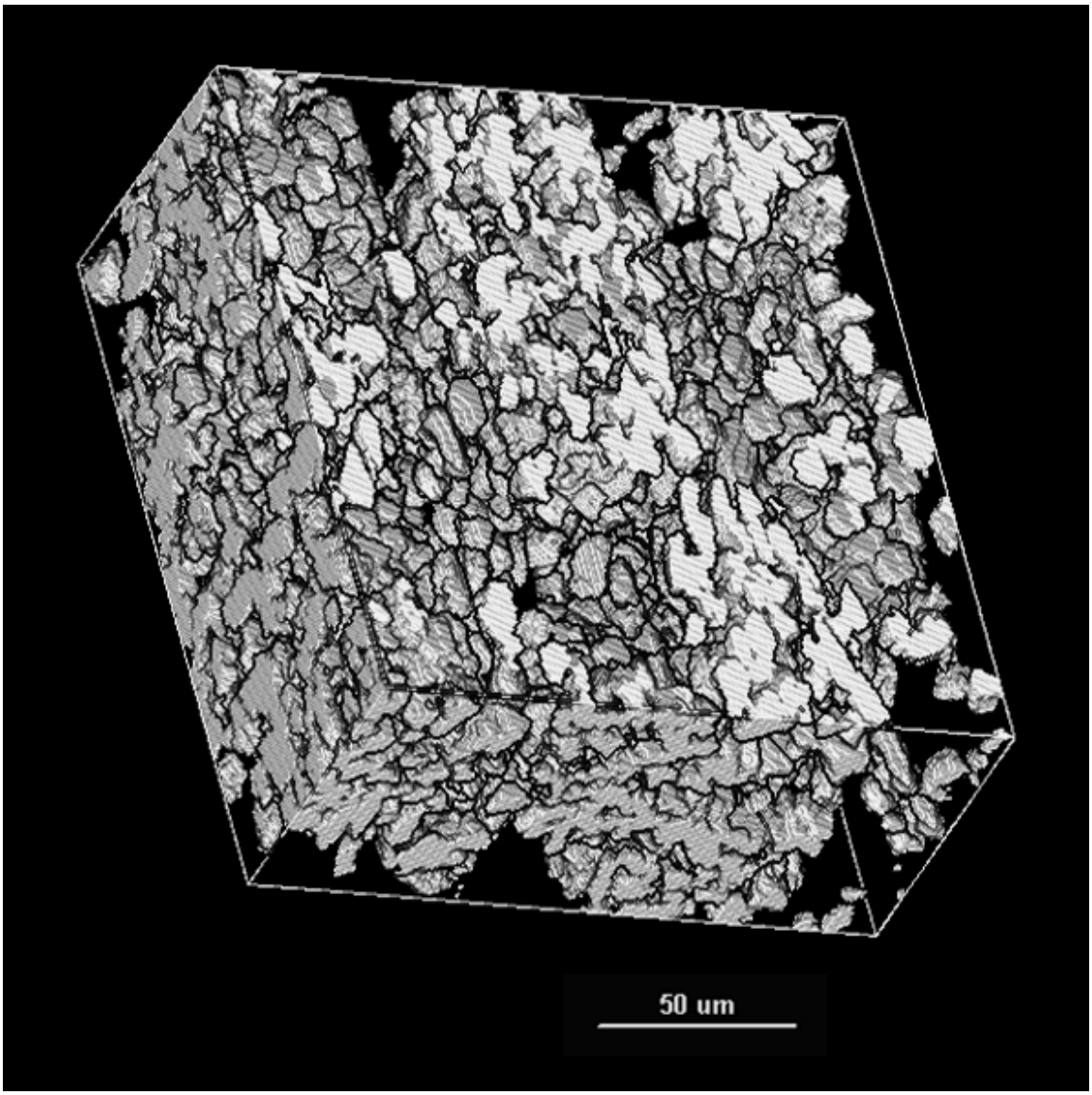


Figure 4.16: Small segment of simulated 3D microstructure of 8.1 PSR DRA composite

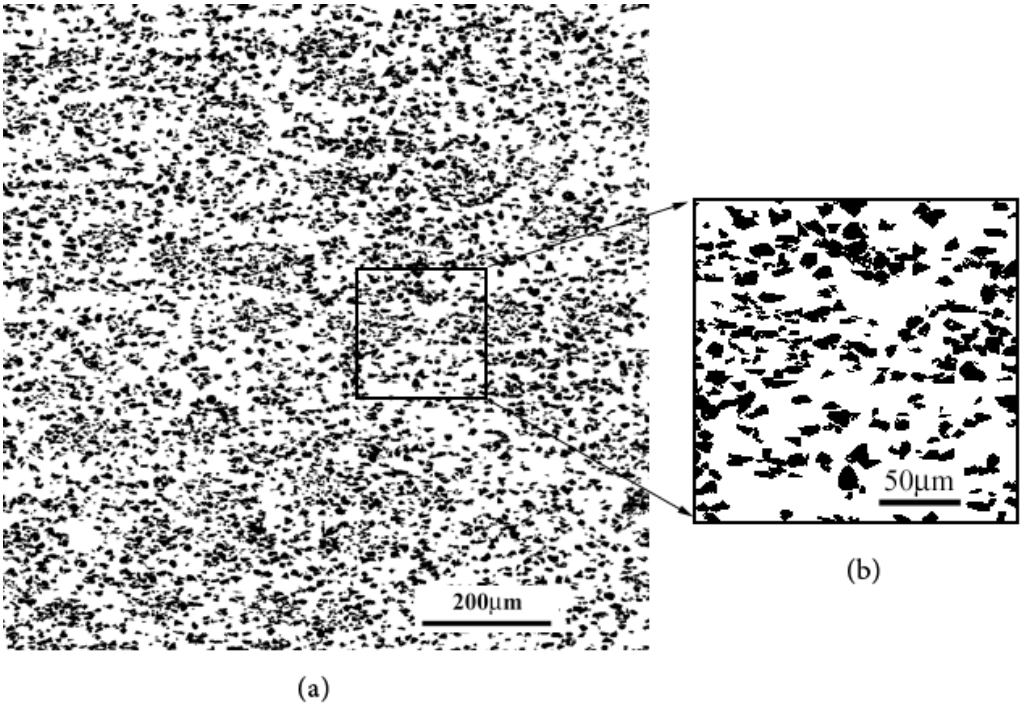


Figure 4.17: (a) 2D montage serial section of 2.0 PSR DRA simulated microstructure. (b) Magnified view of the outlined region in (a).

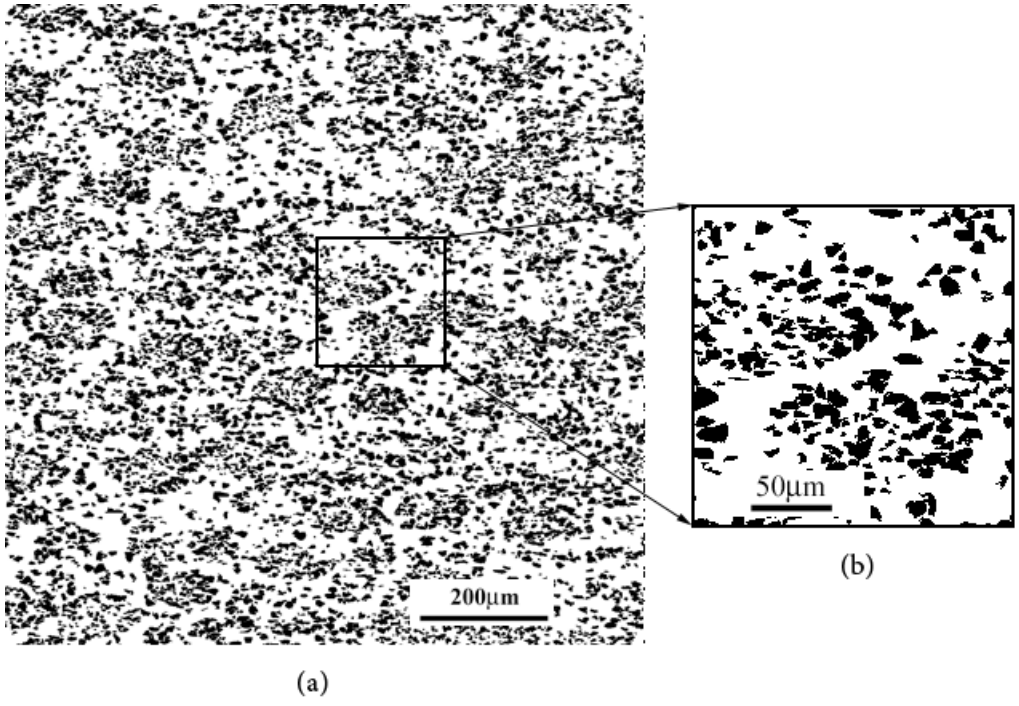


Figure 4.18: (a) 2D montage serial section of 3.1 PSR DRA simulated microstructure. (b) Magnified view of the outlined region in (a).

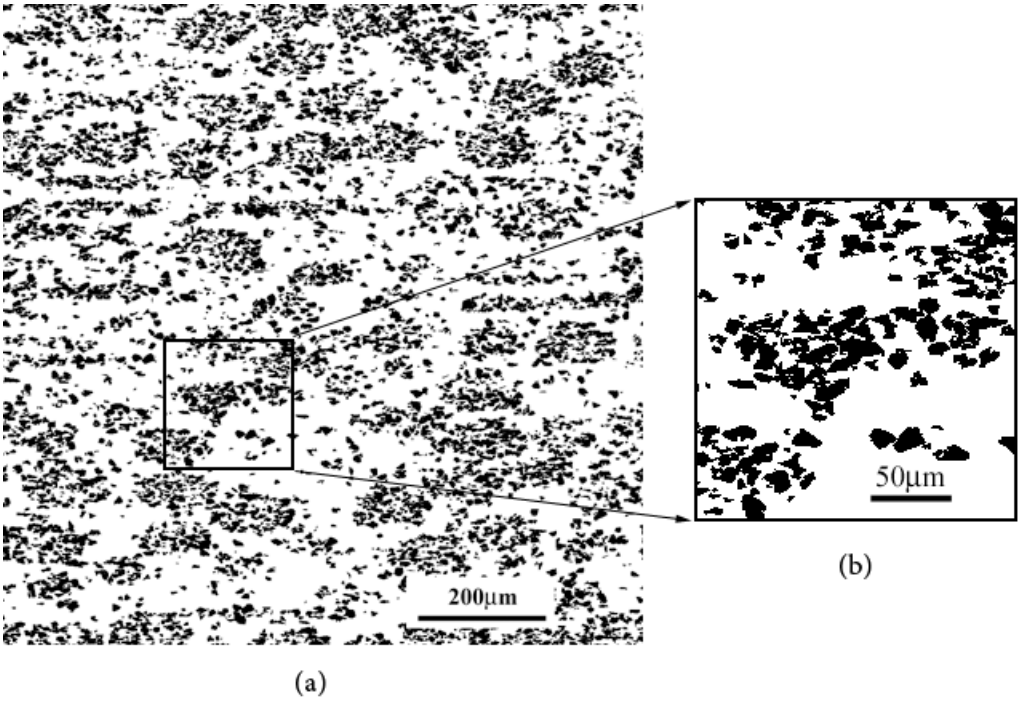


Figure 4.19: (a) 2D montage serial section of 8.1 PSR DRA simulated microstructure. (b) Magnified view of the outlined region in (a).

All simulated DRA's global volume fraction of SiC particles are 0.28, which are the same as in the real microstructures. The values of other simulation parameters are given in Table 4.1.

Table 4.1: Simulation parameters for 3D simulations of DRA composites

PSR	Cluster Type-1			Cluster Type-2			Cluster Intensity	Particle Overlap
	Number Density (mm^{-3})	Major Axis Length (μm)	Minor Axis Length (μm)	Number Density (mm^{-3})	Major Axis Length (μm)	Minor Axis Length (μm)		
2.0	870	270	30	950	130	70	1.48	0.10
3.1	870	270	30	950	130	70	1.65	0.12
8.1	870	270	30	950	130	70	1.92	0.23

The realistic nature of simulated microstructures has been ensured via comparisons to the real microstructures using two-point correlation functions. These functions of the real and simulated microstructures in three perpendicular directions (Figures 4.20 to 4.28) show an excellent match. Short range matching of the functions confirms the similarity of particle sizes and shapes, whereas matching of the long range part of the correlation functions ensures statistical similarity of the long-range heterogeneity of spatial distributions of particles in the real and simulated microstructures.

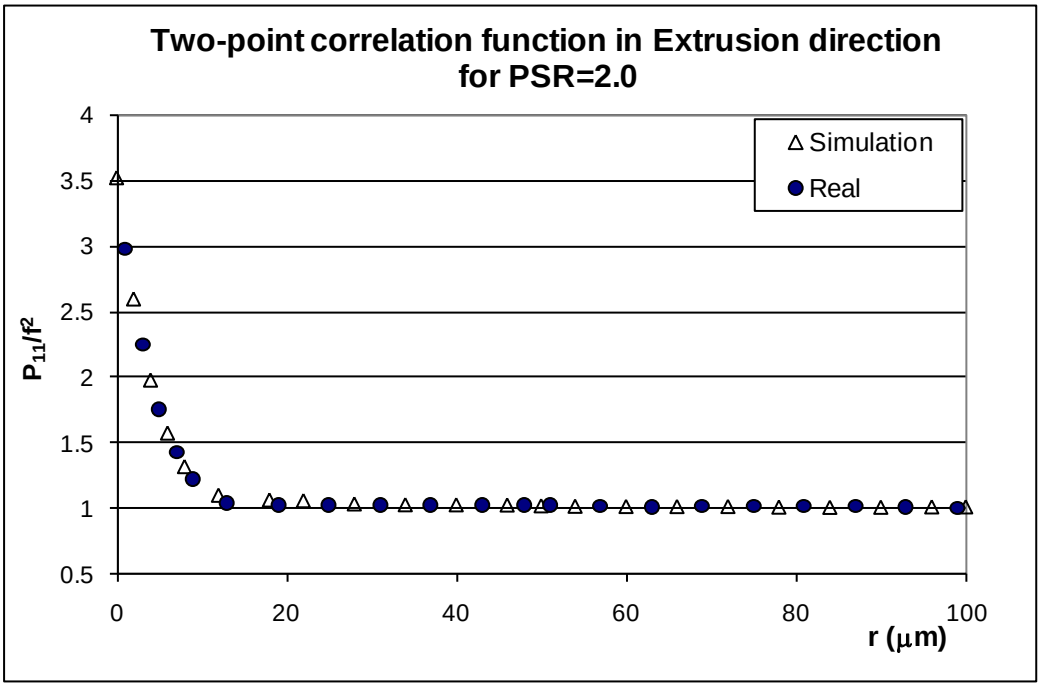


Figure 4.20: Comparison of two-point function in extrusion direction (Z) for 2.0 PSR DRA real and simulated microstructures

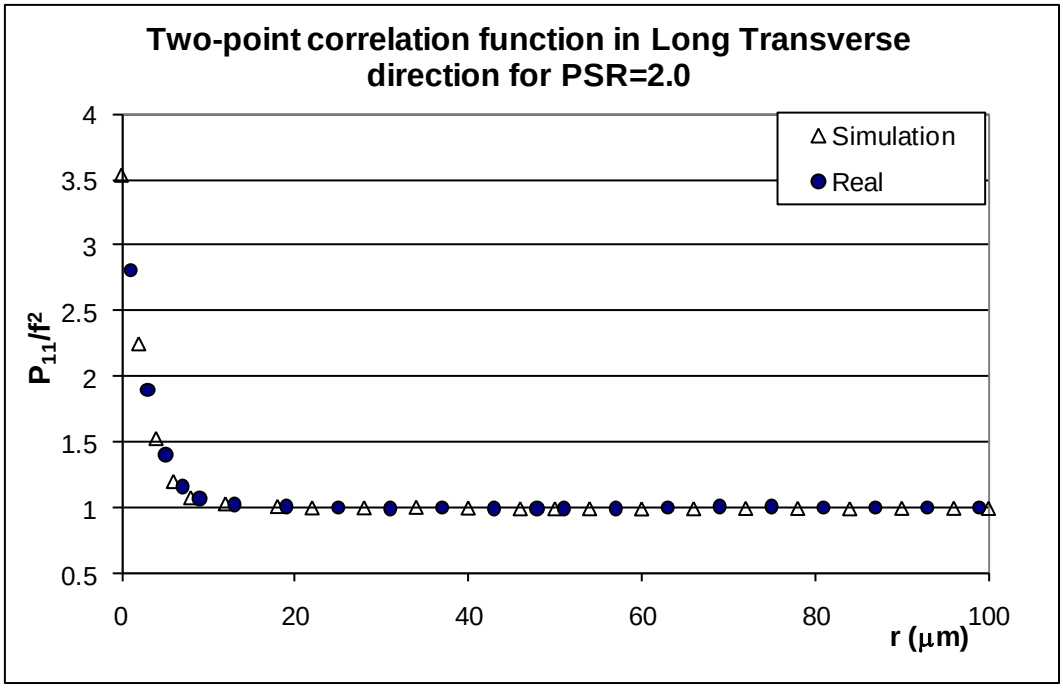


Figure 4.21: Comparison of two-point function in long transverse direction for 2.0 PSR DRA real and simulated microstructures

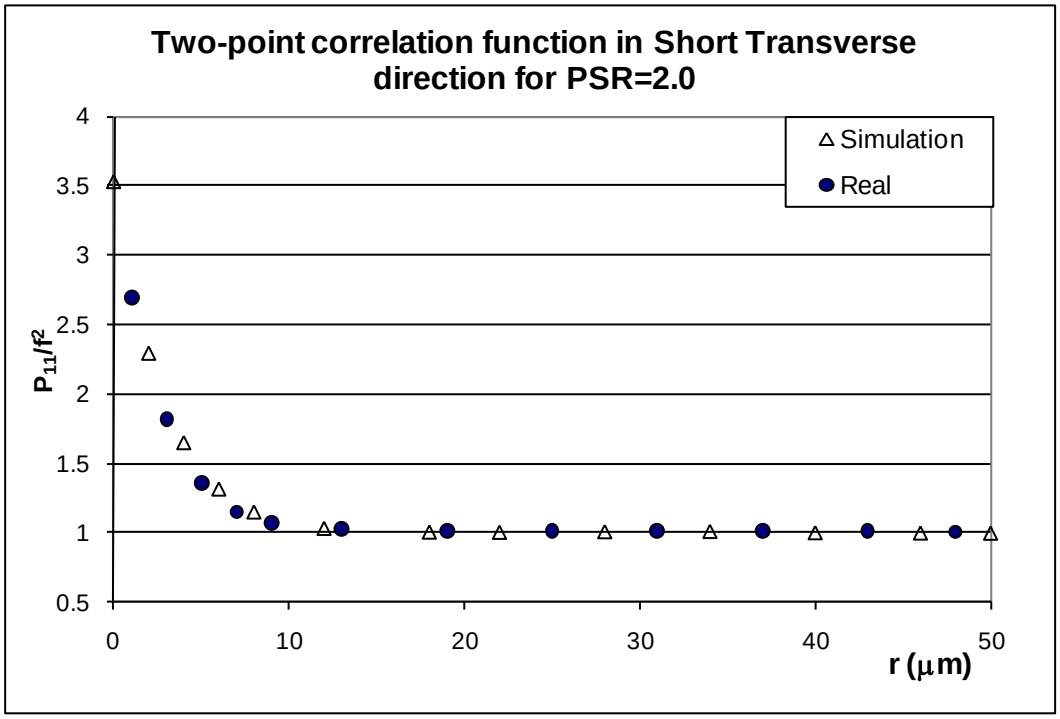


Figure 4.22: Comparison of two-point function in short transverse direction for 2.0 PSR DRA real and simulated microstructures

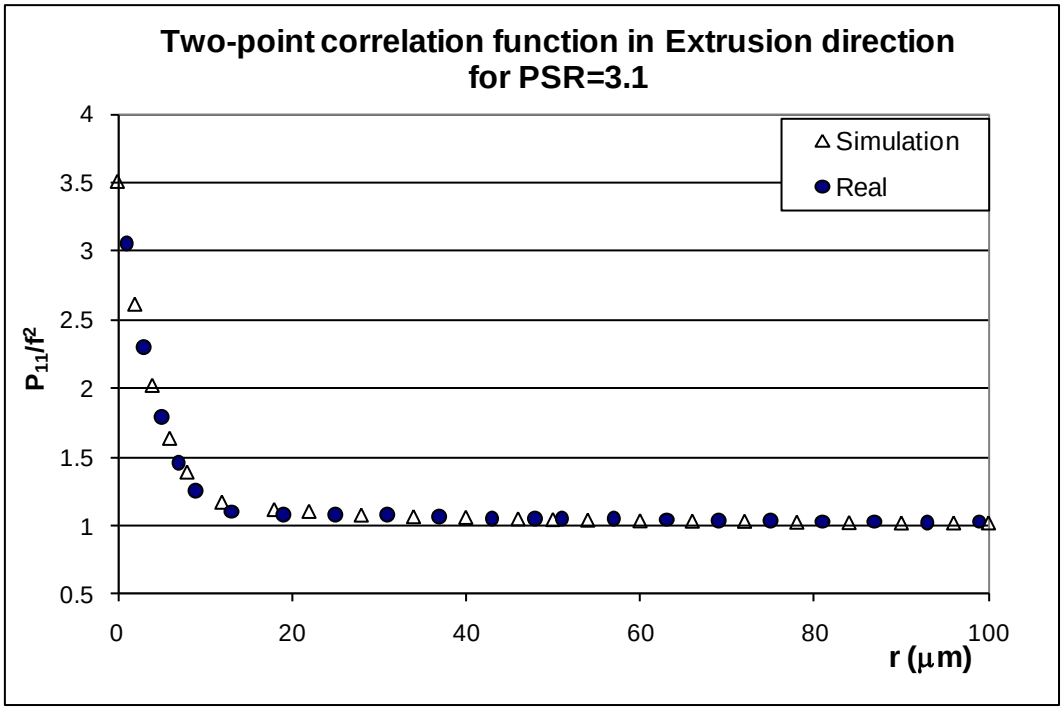


Figure 4.23: Comparison of two-point function in extrusion direction (Z) for 3.1 PSR DRA real and simulated microstructures

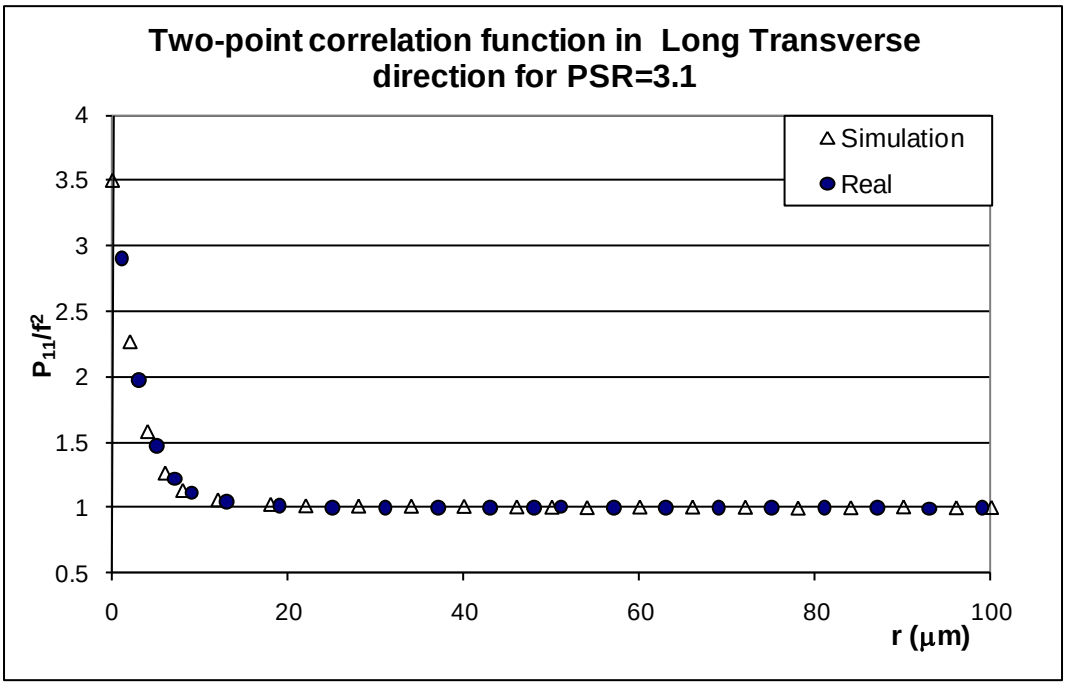


Figure 4.24: Comparison of two-point function in long transverse direction for 3.1 PSR DRA real and simulated microstructures

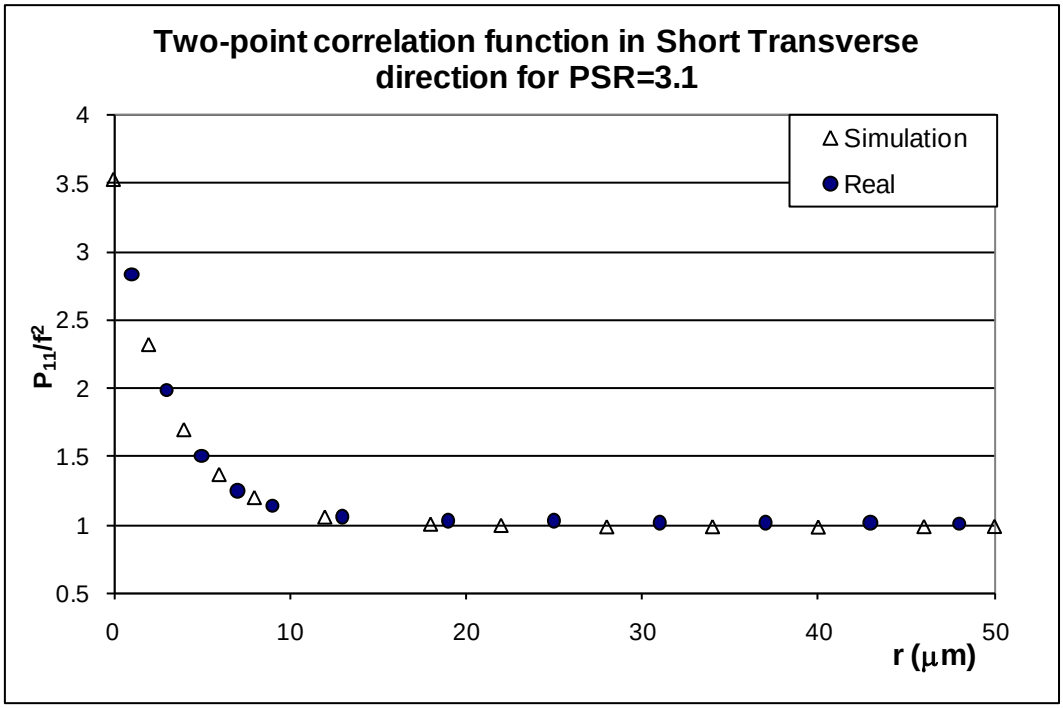


Figure 4.25: Comparison of two-point function in short transverse direction for 3.1 PSR DRA real and simulated microstructures

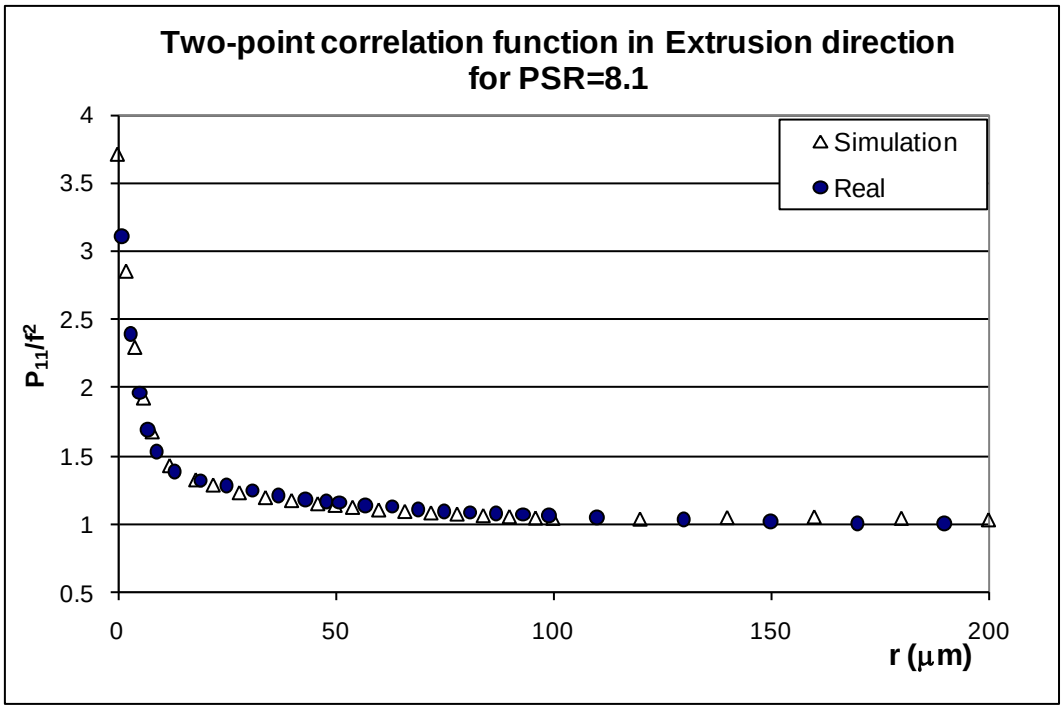


Figure 4.26: Comparison of two-point function in extrusion direction (Z) for 8.1 PSR DRA real and simulated microstructures

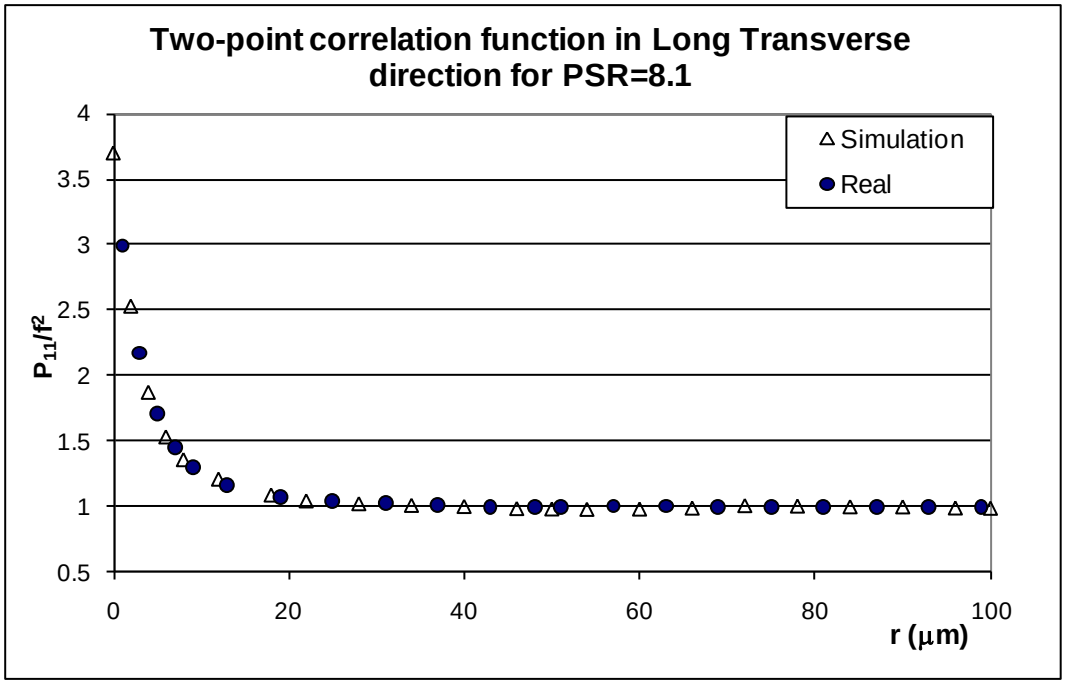


Figure 4.27: Comparison of two-point function in long transverse direction for 8.1 PSR DRA real and simulated microstructures

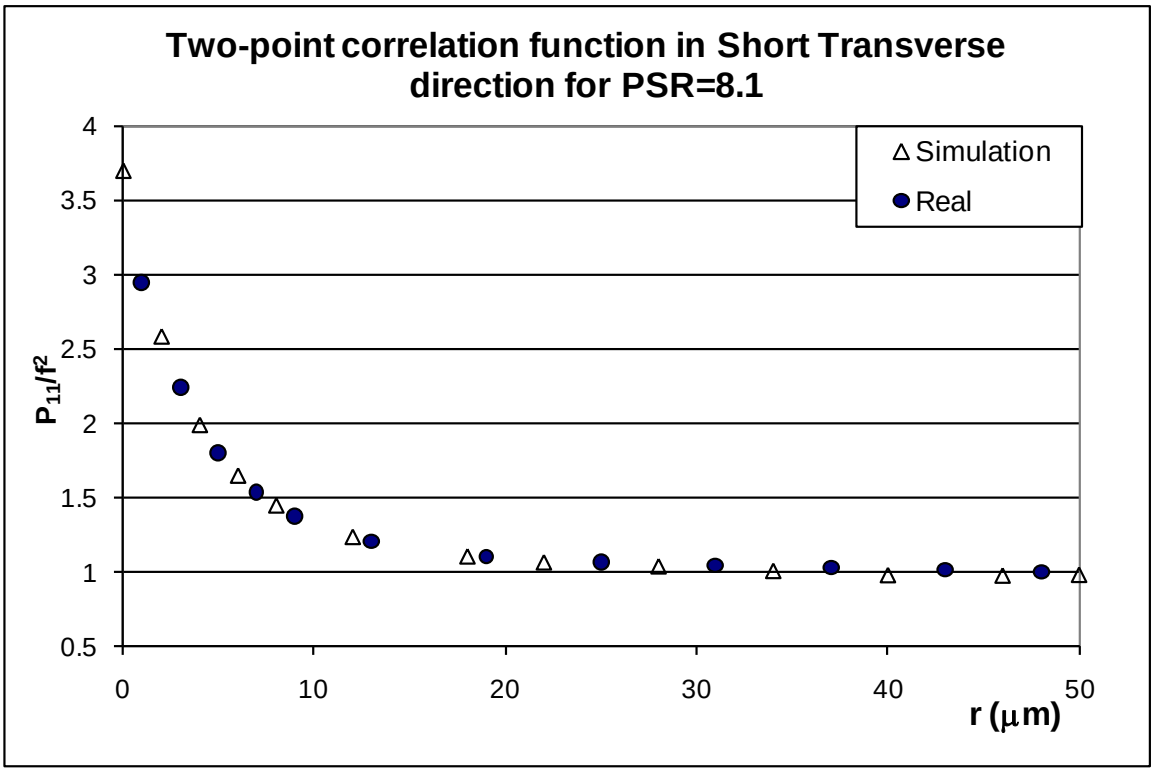
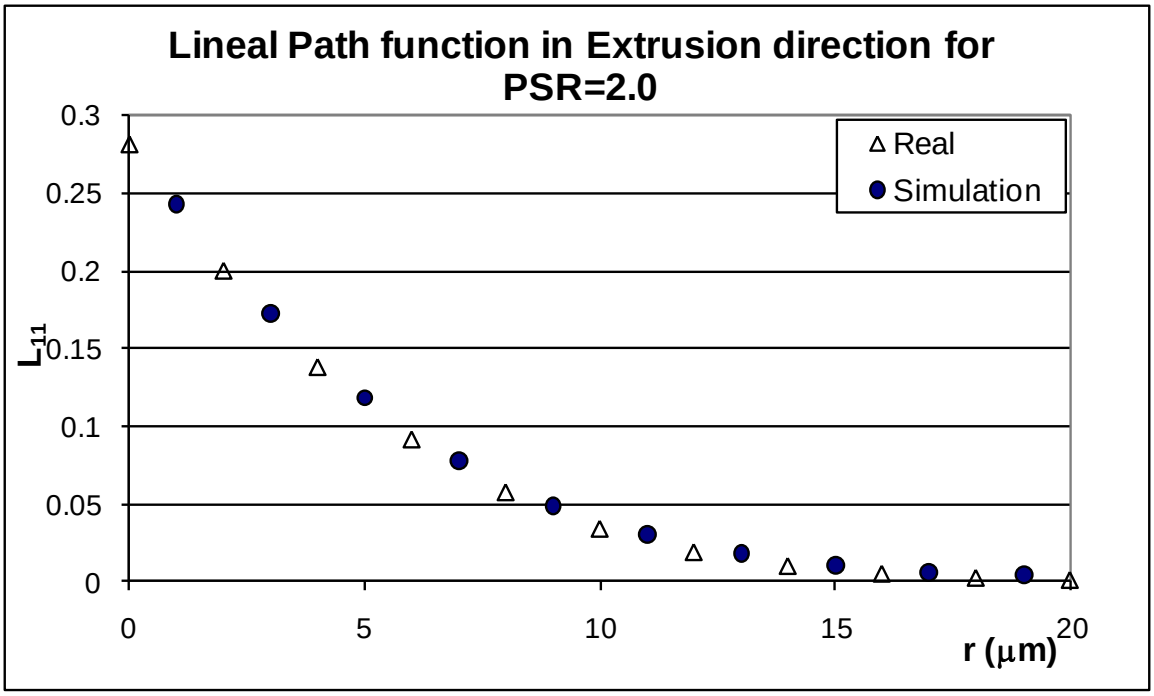
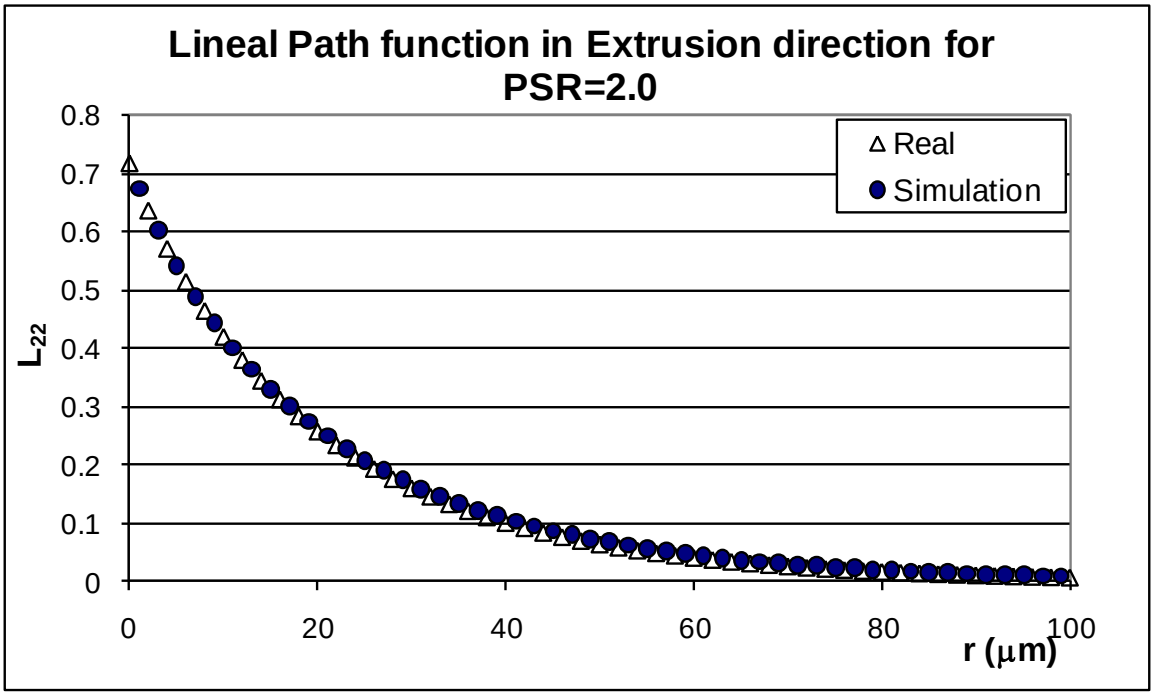


Figure 4.28: Comparison of two-point function in short transverse direction for 8.1 PSR DRA real and simulated microstructures

As reported in Chapter 2, the lineal path probability distributions and two-point correlations functions are in general independent of one another (i.e., a lineal path probability distribution cannot be computed from a given two-point correlation function and vice versa, in general). Therefore, the realistic nature of simulated microstructures can be further validated independently via comparisons to the real microstructures using lineal path probability functions. The lineal path probability distribution functions of the real and simulated microstructures in three perpendicular directions (see Figures 4.29 to 4.37) show an excellent match which validates the statistical similarity of these microstructures. The computer code for measuring lineal path probability distribution is given in Appendix B.2.

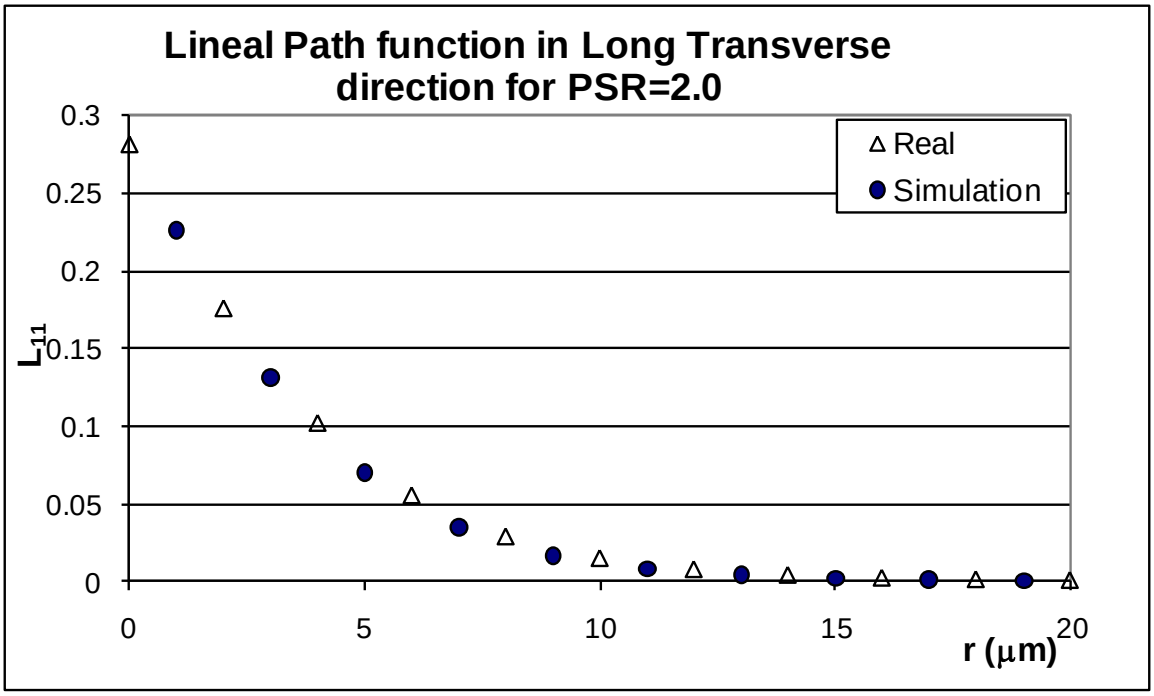


(a)

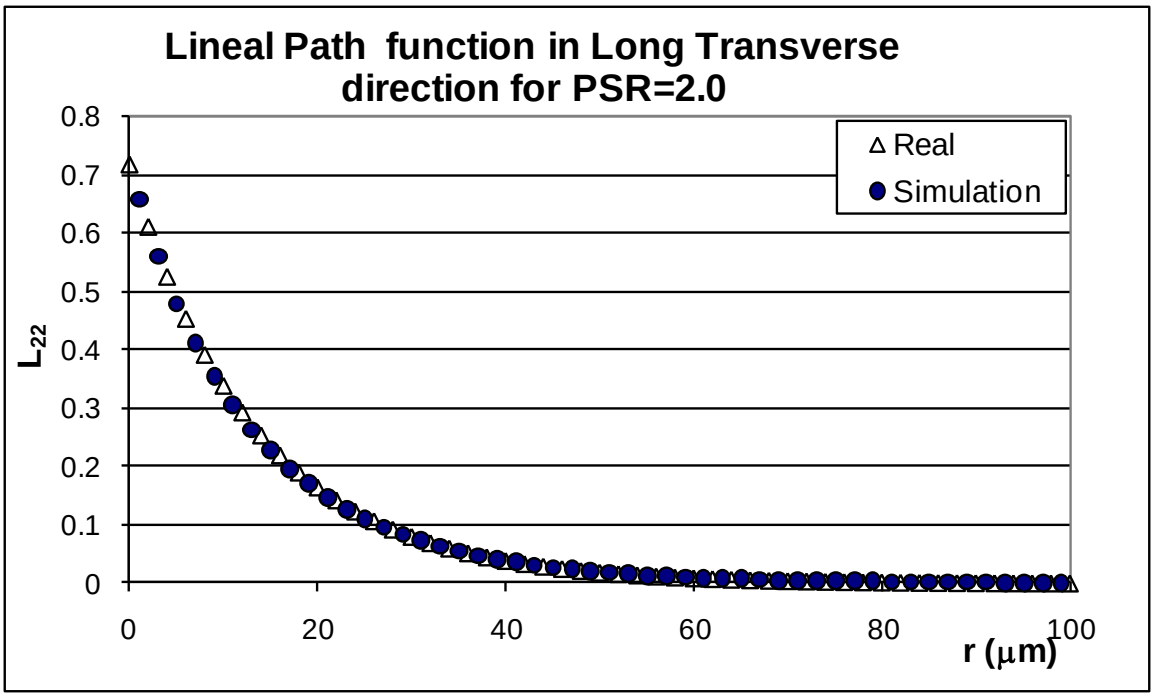


(b)

Figure 4.29: Comparison of lineal path function in extrusion direction for 2.0 PSR DRA real and simulated microstructures. (a) Test lines completely in SiC particles, (b) Test line completely in Al matrix.

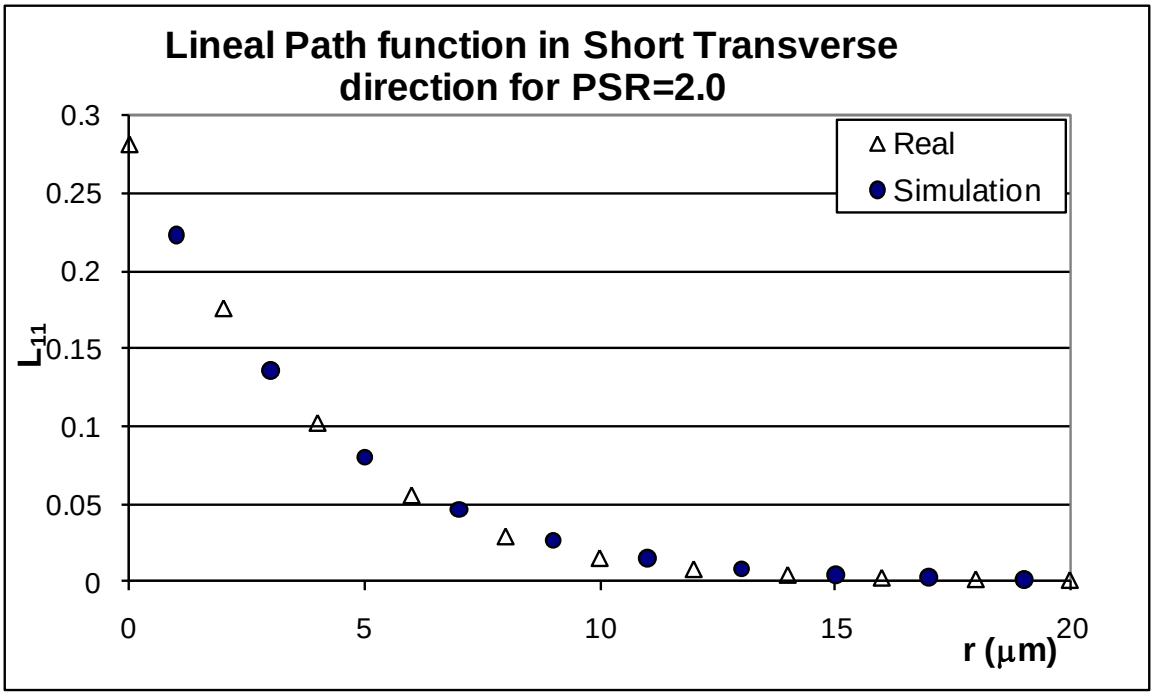


(a)

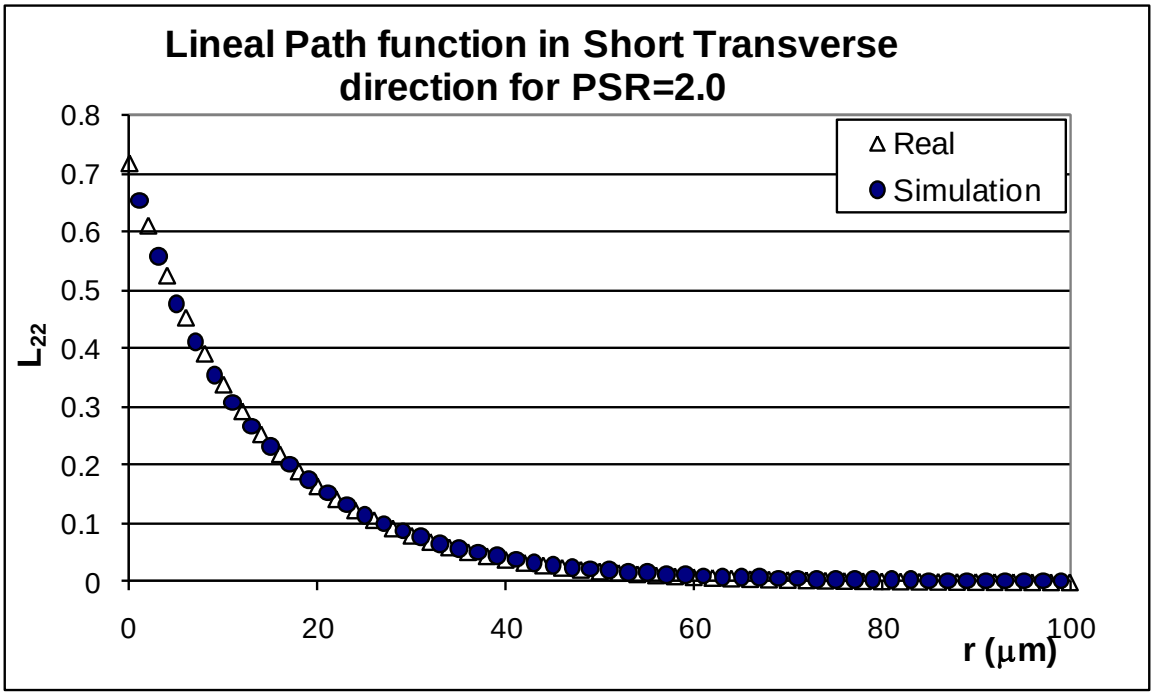


(b)

Figure 4.30: Comparison of lineal path function in long transverse direction for 2.0 PSR DRA real and simulated microstructures. (a) Test lines completely in SiC particles, (b) Test line completely in Al matrix.

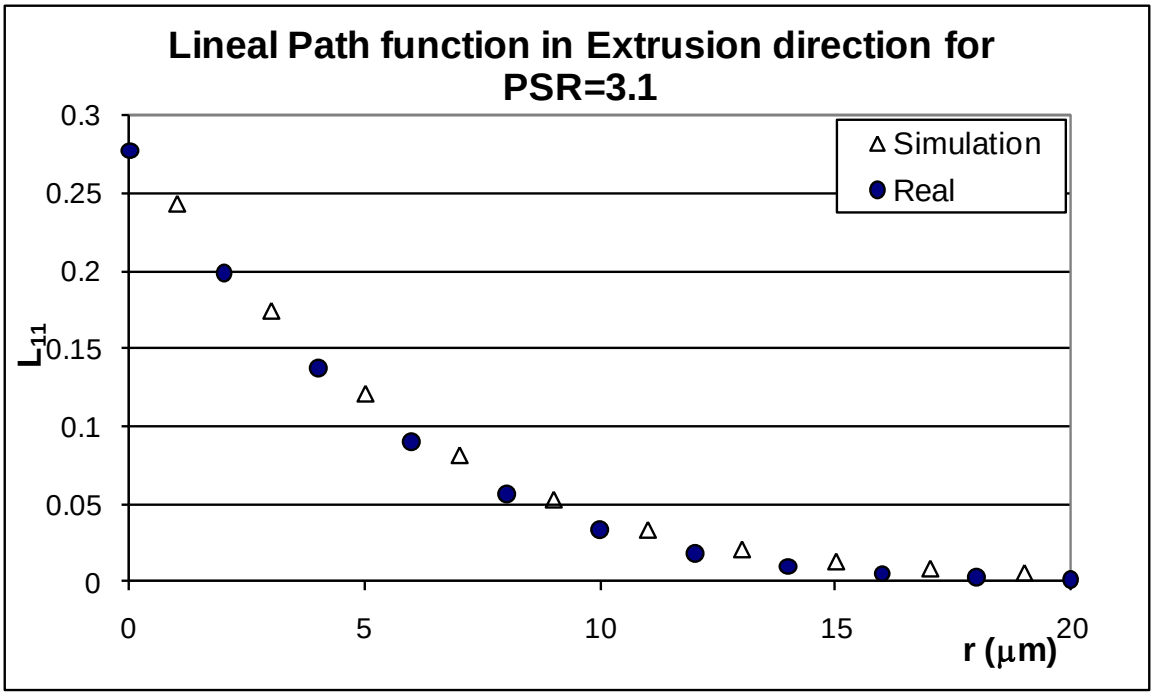


(a)

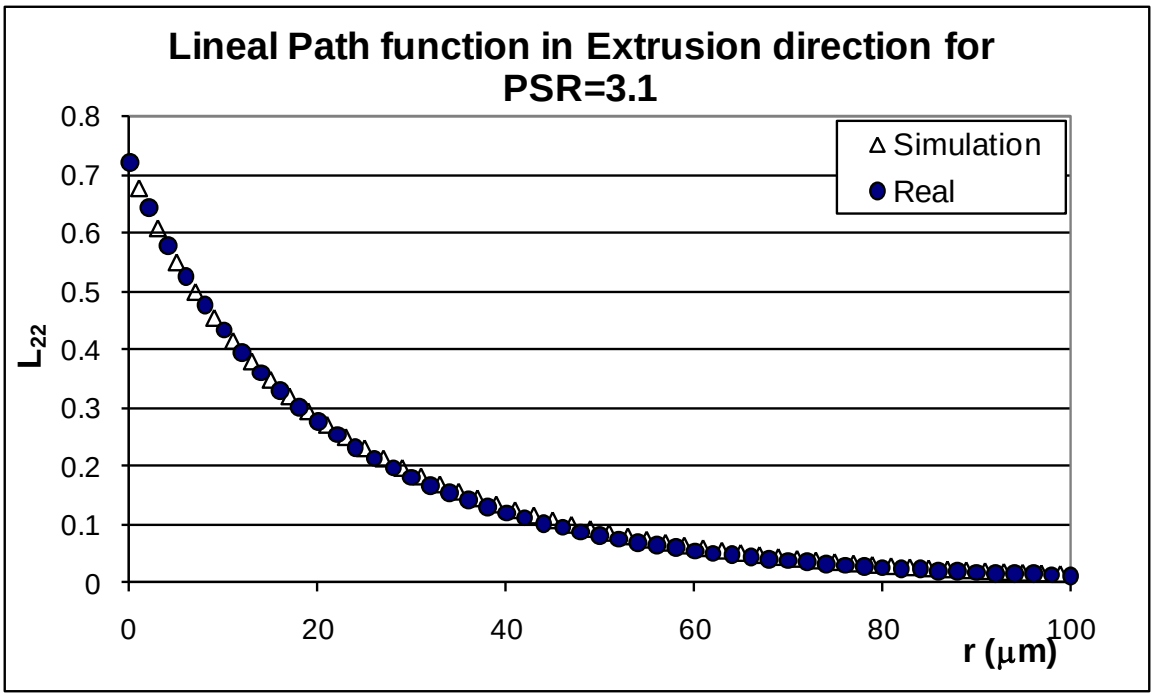


(b)

Figure 4.31: Comparison of lineal path function in short transverse direction for 2.0 PSR DRA real and simulated microstructures. (a) Test lines completely in SiC particles, (b) Test line completely in Al matrix.

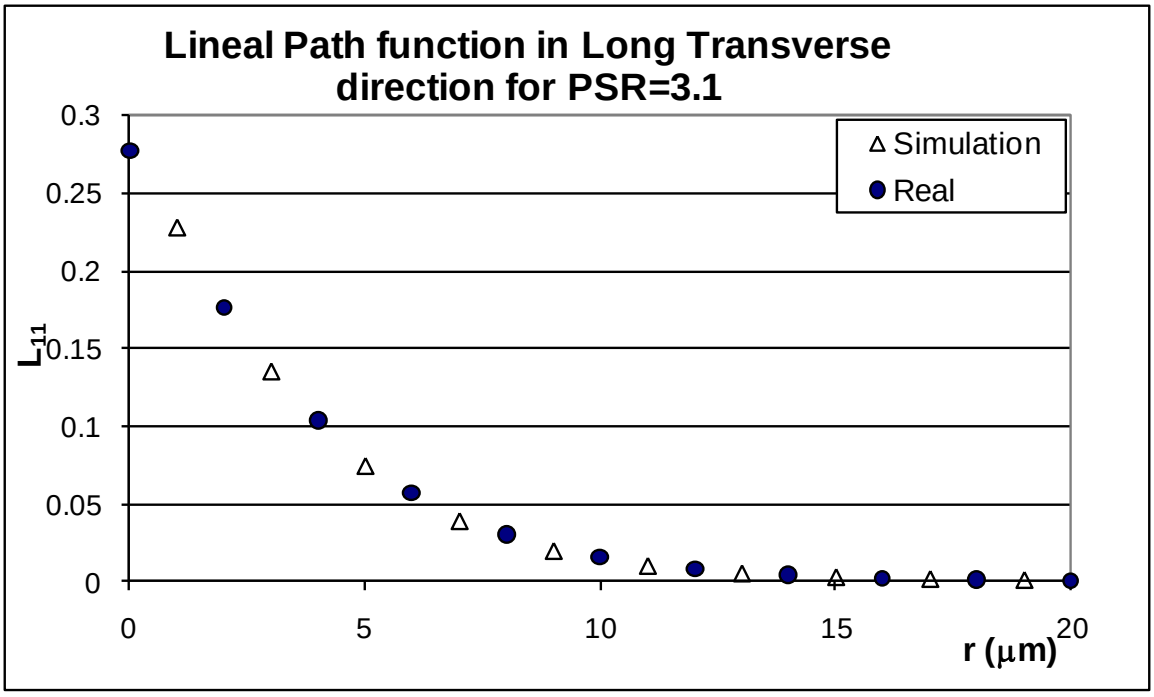


(a)

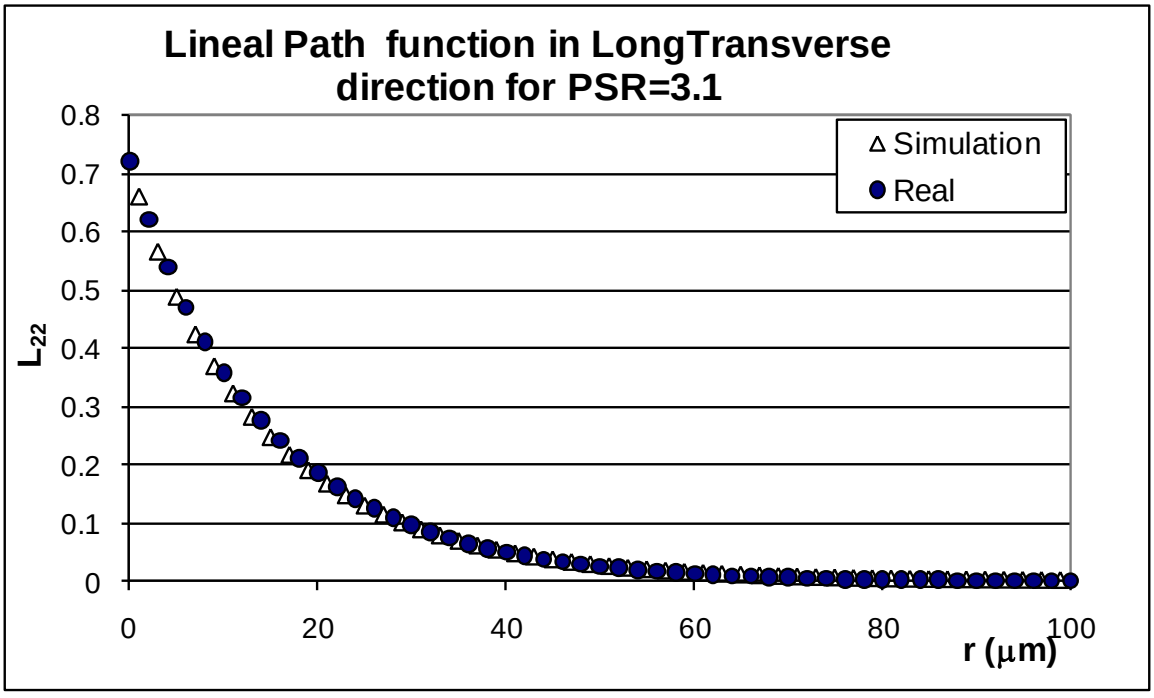


(b)

Figure 4.32: Comparison of lineal path function in extrusion direction for 3.1 PSR DRA real and simulated microstructures. (a) Test lines completely in SiC particles, (b) Test line completely in Al matrix.

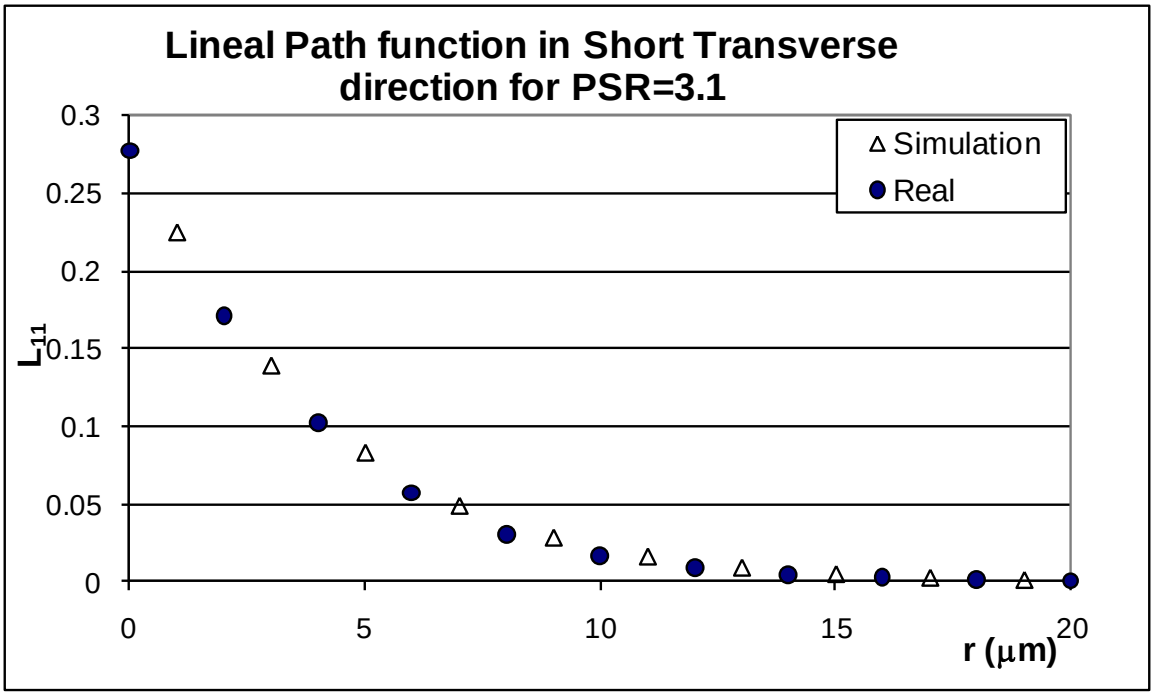


(a)

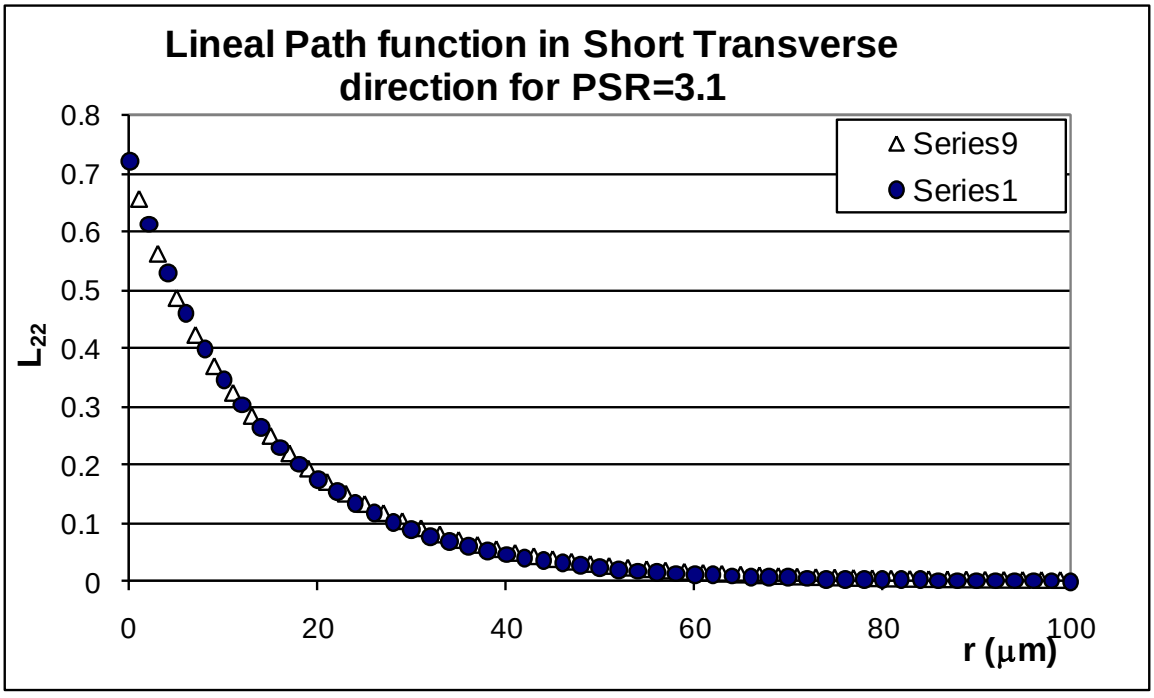


(b)

Figure 4.33: Comparison of lineal path function in long transverse direction for 3.1 PSR DRA real and simulated microstructures. (a) Test lines completely in SiC particles, (b) Test line completely in Al matrix.

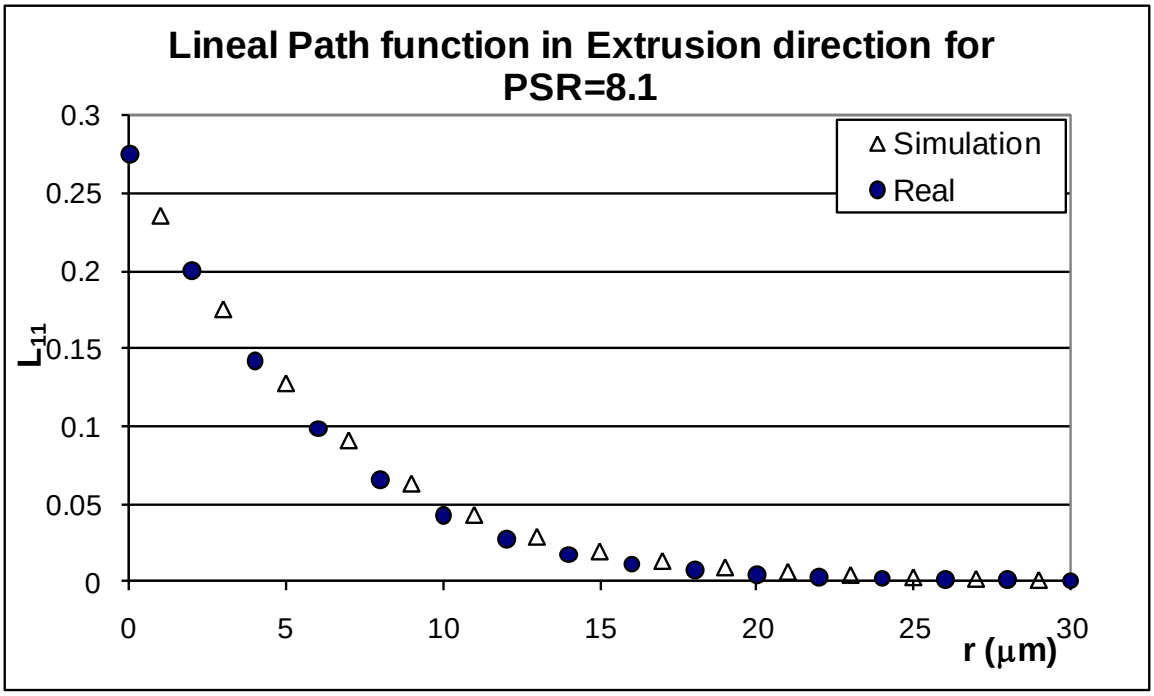


(a)

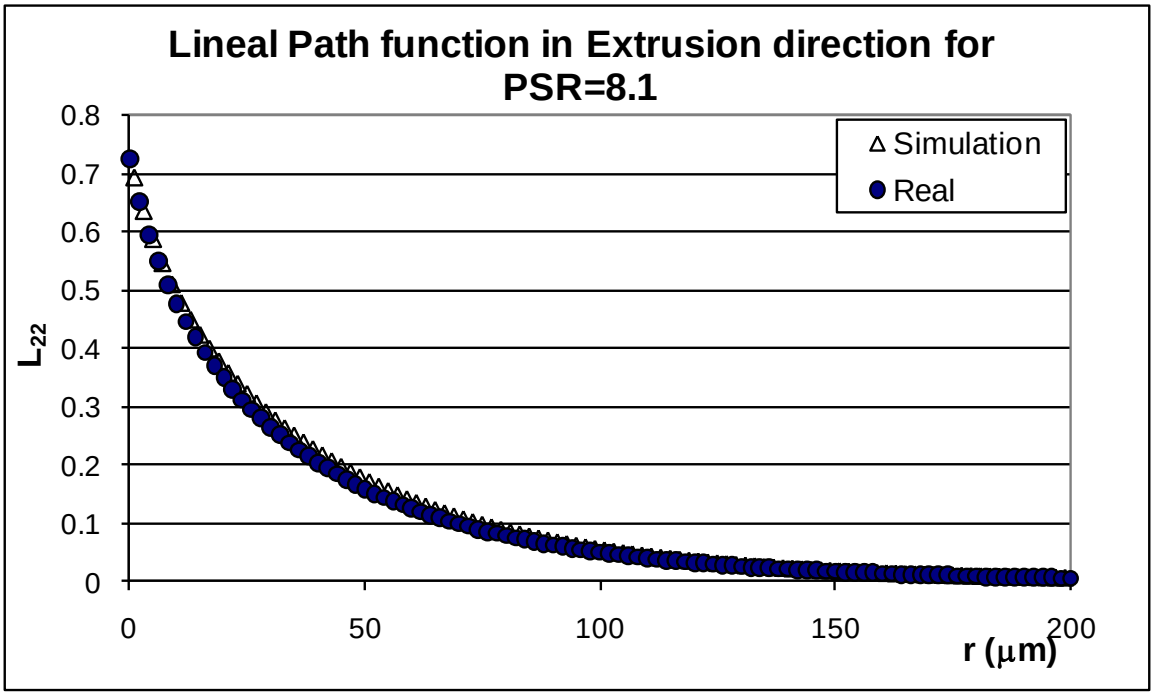


(b)

Figure 4.34: Comparison of lineal path function in short transverse direction for 3.1 PSR DRA real and simulated microstructures. (a) Test lines completely in SiC particles, (b) Test line completely in Al matrix.

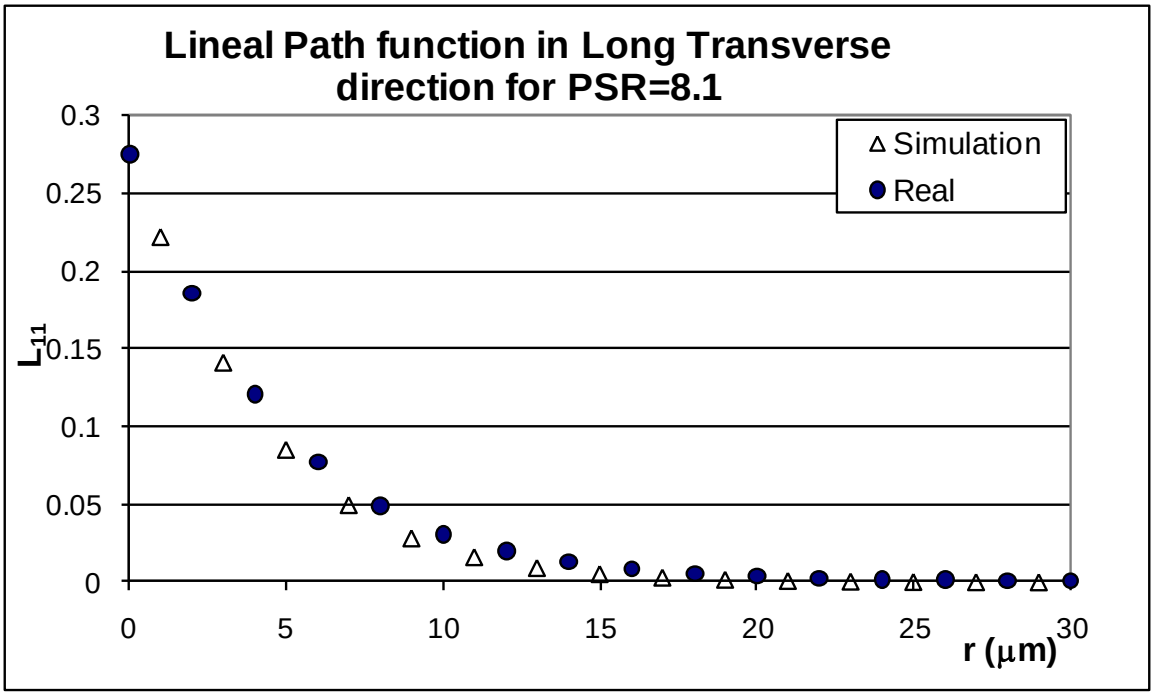


(a)

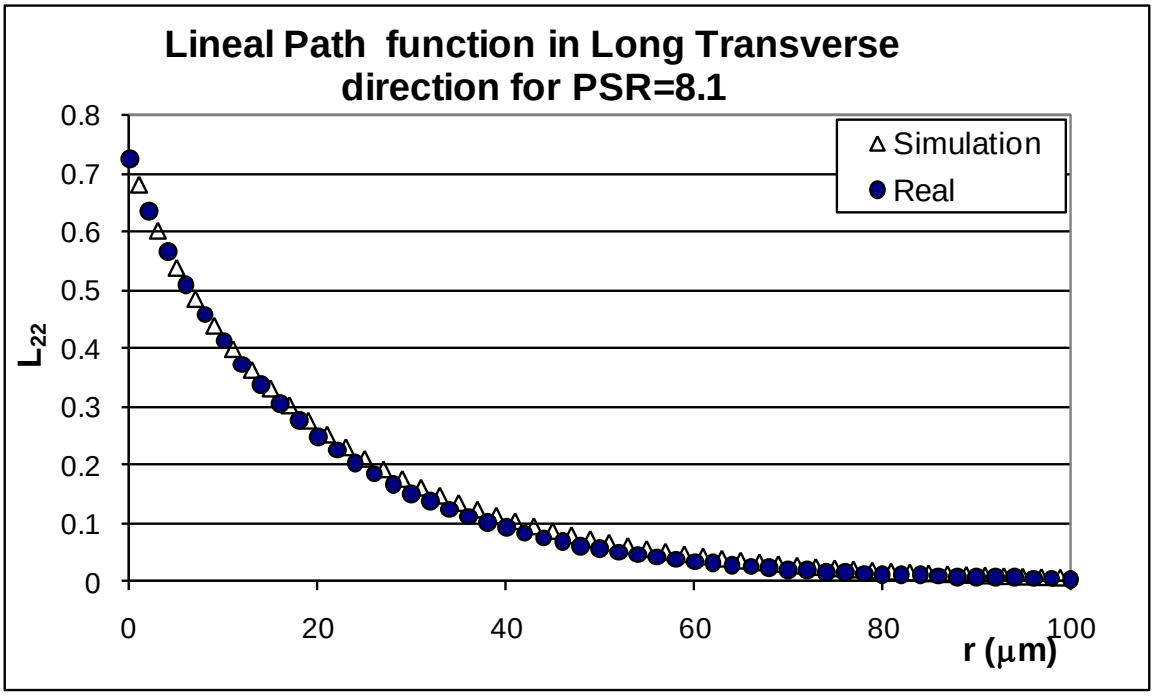


(b)

Figure 4.35: Comparison of lineal path function in extrusion direction for 8.1 PSR DRA real and simulated microstructures. (a) Test lines completely in SiC particles, (b) Test line completely in Al matrix.

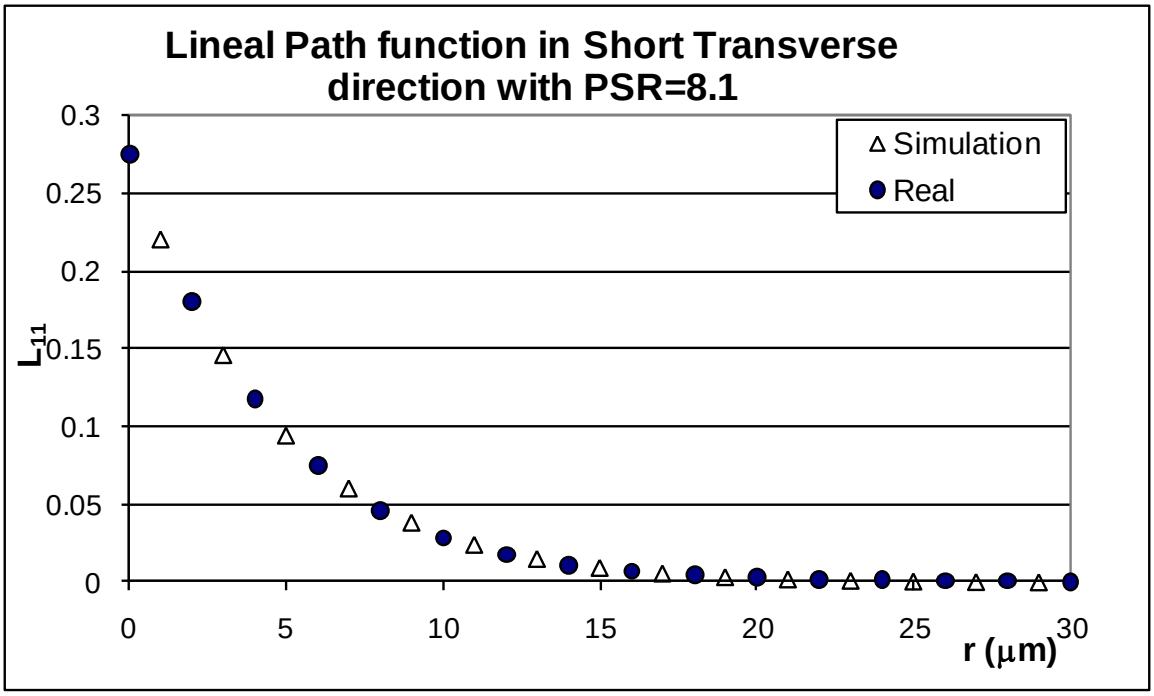


(a)

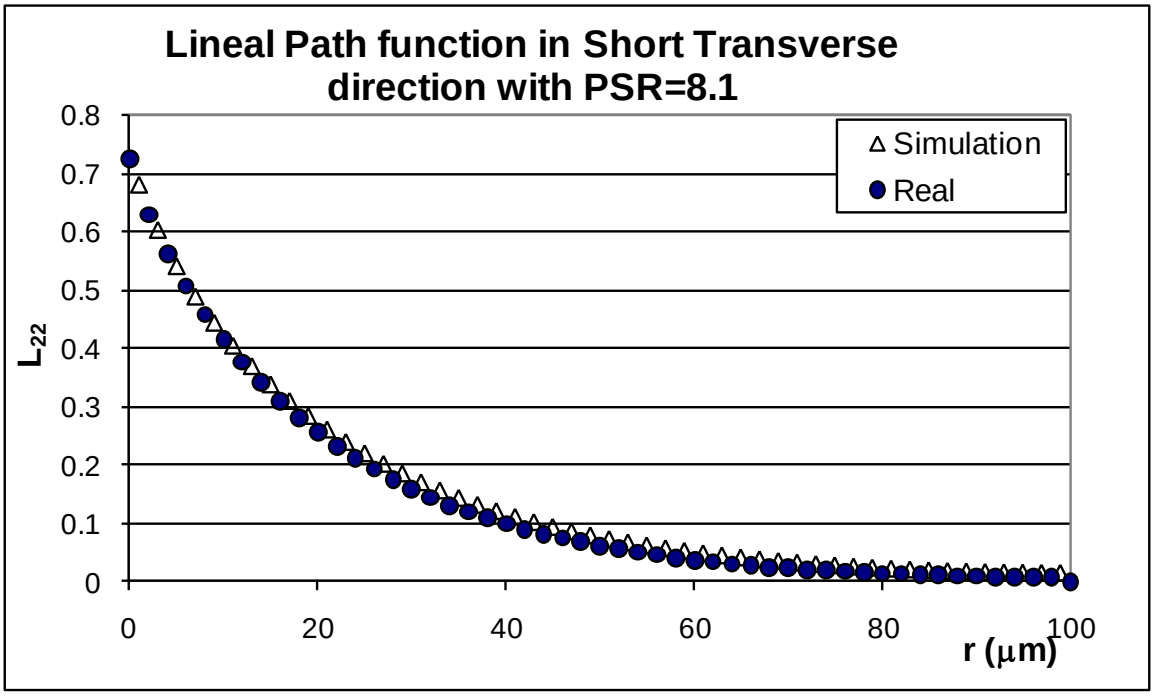


(b)

Figure 4.36: Comparison of lineal path function in long transverse direction for 8.1 PSR DRA real and simulated microstructures. (a) Test lines completely in SiC particles, (b) Test line completely in Al matrix.



(a)



(b)

Figure 4.37: Comparison of lineal path function in short transverse direction for 8.1 PSR DRA real and simulated microstructures. (a) Test lines completely in SiC particles, (b) Test line completely in Al matrix.

A 3D microstructure simulated using present methodology is *realistic* because (i) it incorporates realistic complex 3D particles shapes/morphologies similar to those present in the corresponding real microstructures, (ii) it incorporates spatial clustering of SiC particles and anisotropy, as represented by two-point correlation function, similar to those in the corresponding real microstructures, (iii) it incorporates same volume fraction and size distribution of the SiC particles same as those in the corresponding real microstructure, (iv) the simulated microstructure is sufficiently large (more than 70,000 SiC particles covering about 10^8 μm^3 3D volume) and has sufficiently small voxel size (1 $\times 1 \times 1$ μm), so that the short-range (0 to 10 μm), intermediate-range (10 to 50 μm), and long-range (50 to 100 μm) spatial patterns and other microstructural details are represented at high resolution, and (v) the simulated microstructure is generated by matching their two-point correlation functions along various different directions (angle θ and ϕ) with the corresponding experimental data to ensure that spatial patterns along all directions are correctly represented, (vi) the statistical similarity of simulated and corresponding real microstructures is validated by the lineal path probability distribution functions. The present simulation procedure is sufficiently flexible so that any specified extent of overlap can be permitted between the SiC particles, and if needed, the particles can be rotated to any specified extent to simulate any desired morphological anisotropy.

4.2.6 Computer Simulated Virtual 3D DRA Microstructures

The same methodology can be used to create 'virtual' microstructures of the composites that have not yet been fabricated by varying the numerical parameters in the model. Let us consider an example of how the simulation parameters can be correlated to the process conditions. Inspection of Table I reveals that the three microstructures having

different spatial clustering of SiC particles due to different PSR (a process parameter) values have been simulated by changing the value of two simulation parameters, namely, clustering intensity and particle overlap; all other simulation parameters have the same values for the three simulated microstructures. Therefore, the changes in the simulation parameters, clustering intensity and particle overlap, essentially represent the variations in the microstructure due to the changes in the process parameter, PSR. Figure 4.38 depicts strong correlation between the simulation parameter clustering intensity and the process parameter PSR. Figure 4.39 depicts strong correlation between particle overlap and PSR.

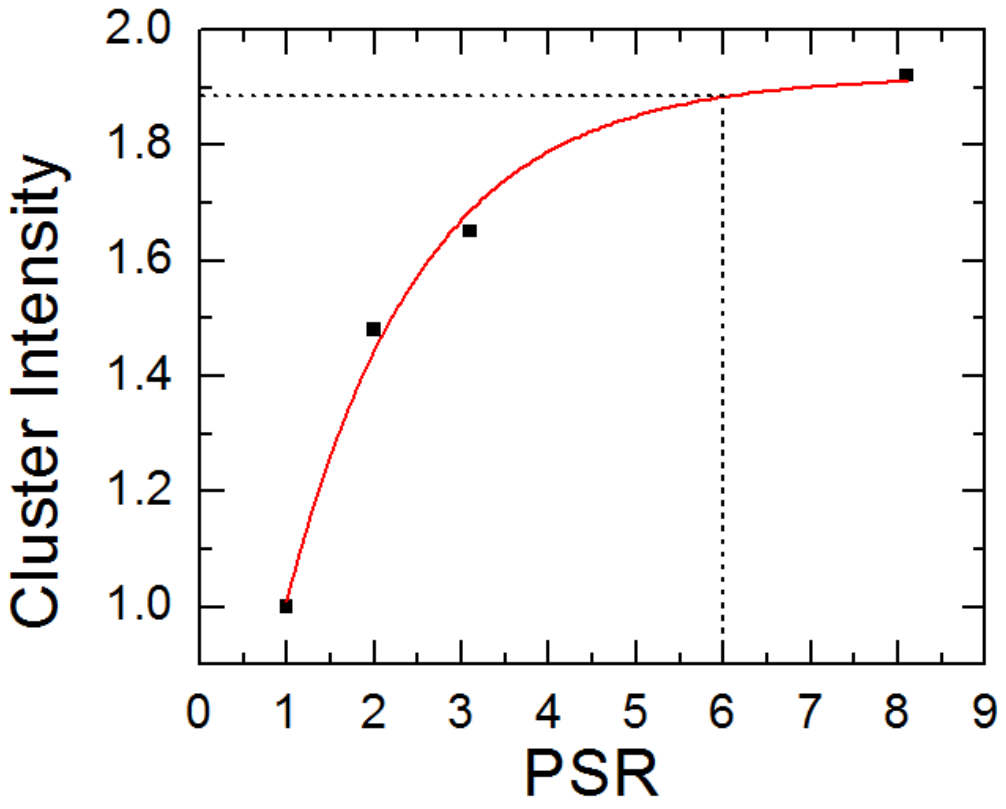


Figure 4.38: Correlation between clustering intensity and PSR

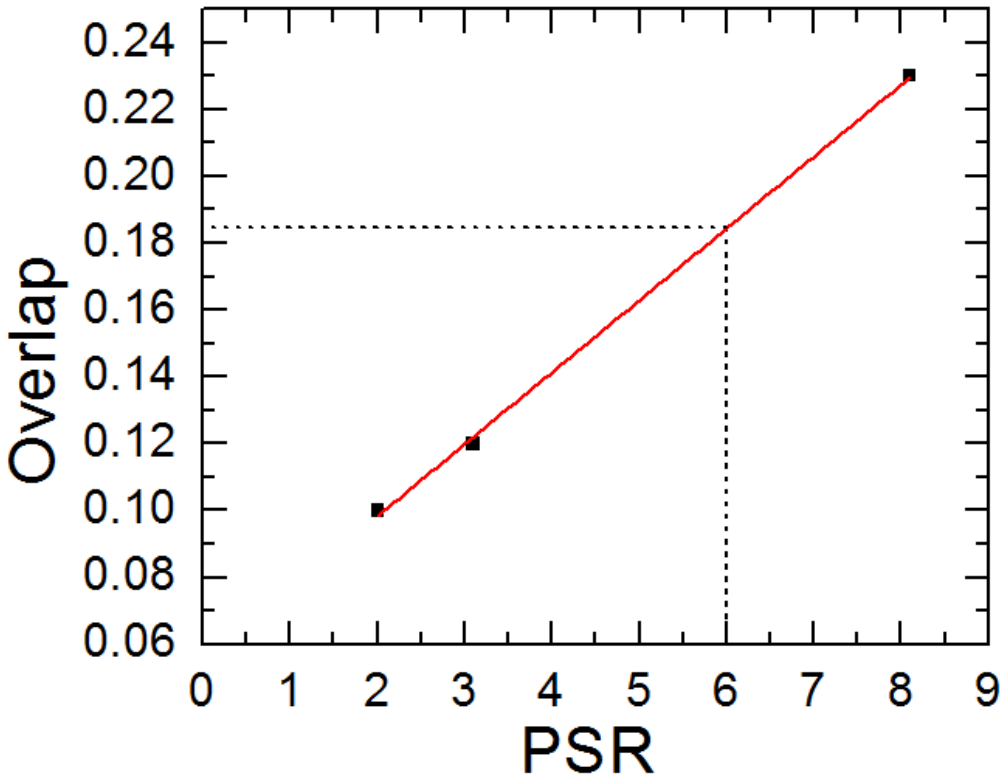


Figure 4.39: Correlation between particle overlap and PSR

One can now utilize these correlations to simulate the virtual 3D microstructures corresponding different PSR values but the same volume fraction and size and shape distribution of SiC particles. Such simulations represent a set of microstructures where all process parameters except PSR have been kept constant. For example, suppose one wants simulate the *virtual* microstructure corresponding to PSR of 6.0. The correlation in Figure 4.38 indicates that the clustering intensity for the composite having 6.0 PSR would be 1.89. Figure 4.39 indicates that the particle overlap would be 0.185. It is now possible to simulate the corresponding microstructure with all *details* using the same simulation model but changing the clustering intensity value to 1.89 and particle overlap value to

0.185. Figure 4.40 shows a small segment of such simulated *virtual* 3D microstructure of the composite having 6.0 PSR and all other process parameters the same as those for the other microstructures (see Table 4.2). Each simulated montage serial section also has a size of 1000 × 1000 pixels at a resolution of 1 μm per pixel (see Figure 16). The microstructure in Figure 4.40 is a virtual 3D microstructure because it was generated without actual fabrication of the corresponding composite. Note that this virtual microstructure has the same realistic 3D SiC particle morphologies, and it incorporates realistic short-range (0 to 10 μm), intermediate-range (10 to 50 μm), and long-range (50 to 500 μm) spatial patterns and microstructural details at high resolution (1 μm pixel size). Similarly, one can simulate virtual microstructures of composites that cover a complete range of PSR values that may be of interest from the experimental data on few (in this case, three) composites having different PSRs.

Table 4.2: Simulation parameters used to simulate 3D microstructure of PSR=6.0 DRA composite. (Volume fraction of SiC particles is 28%)

PSR	Cluster Type-1			Cluster Type-2			Cluster Intensity	Particle Overlap
	Number Density (mm^{-3})	Major Axis Length (μm)	Minor Axis Length (μm)	Number Density (mm^{-3})	Major Axis Length (μm)	Minor Axis Length (μm)		
6.0	870	270	30	950	130	70	1.89	0.185

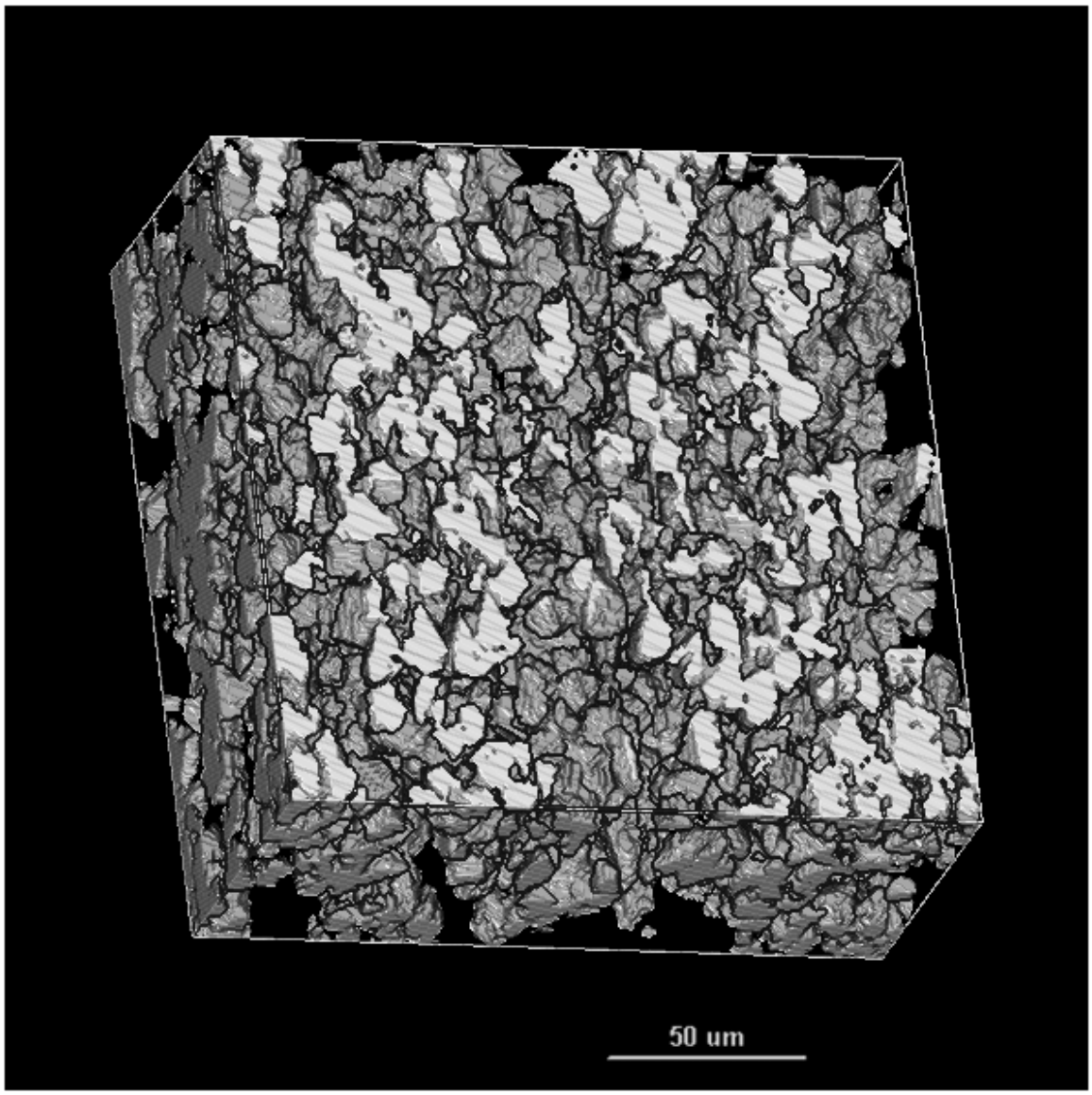


Figure 4.40: Small segment of simulated 3D microstructure of virtual DRA composite with PSR 6.0

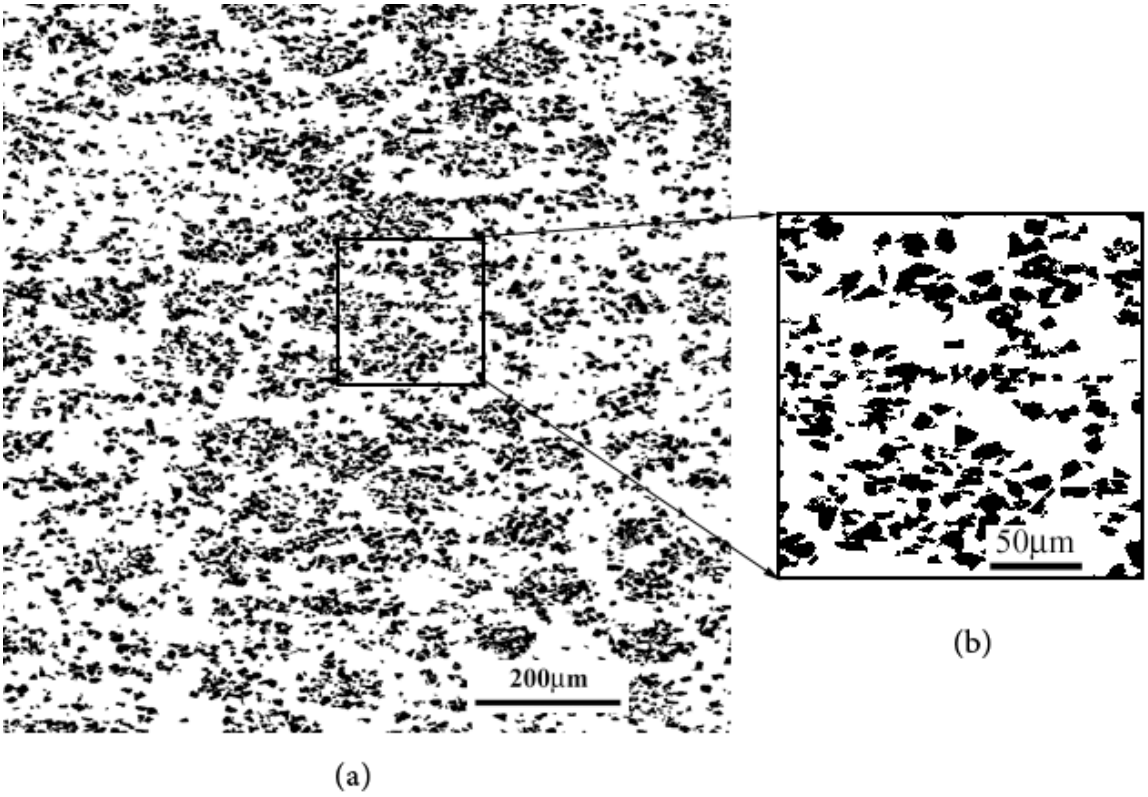


Figure 4.41: (a) Montage of simulated serial section of virtual DRA composite with PSR 6.0 (b) Magnified view of the outlined region in (a)

Figures 4.42 and 4.43 show another example of virtual 3D microstructure simulation, where the spatial arrangement and size/shape distribution of the SiC particles is statistically similar to that in PSR=8.1 DRA composite (see Figure 4.10) but the volume fraction of SiC particles (15%) is lower than that in Figure 4.10 (28%).

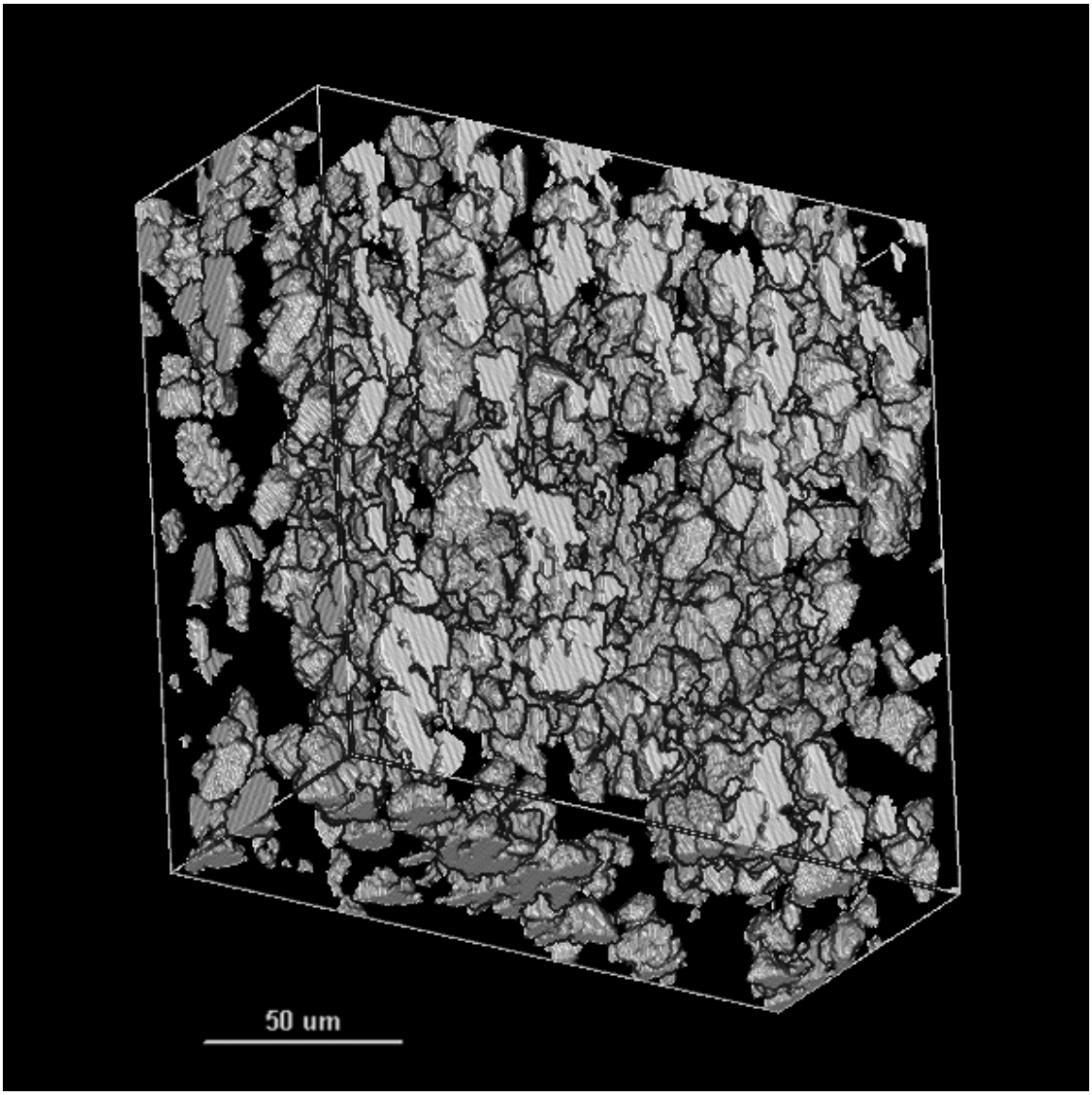


Figure 4.42: Small segment of simulated 3D microstructure with low volume fraction (15%) of SiC particles and PSR=8.1

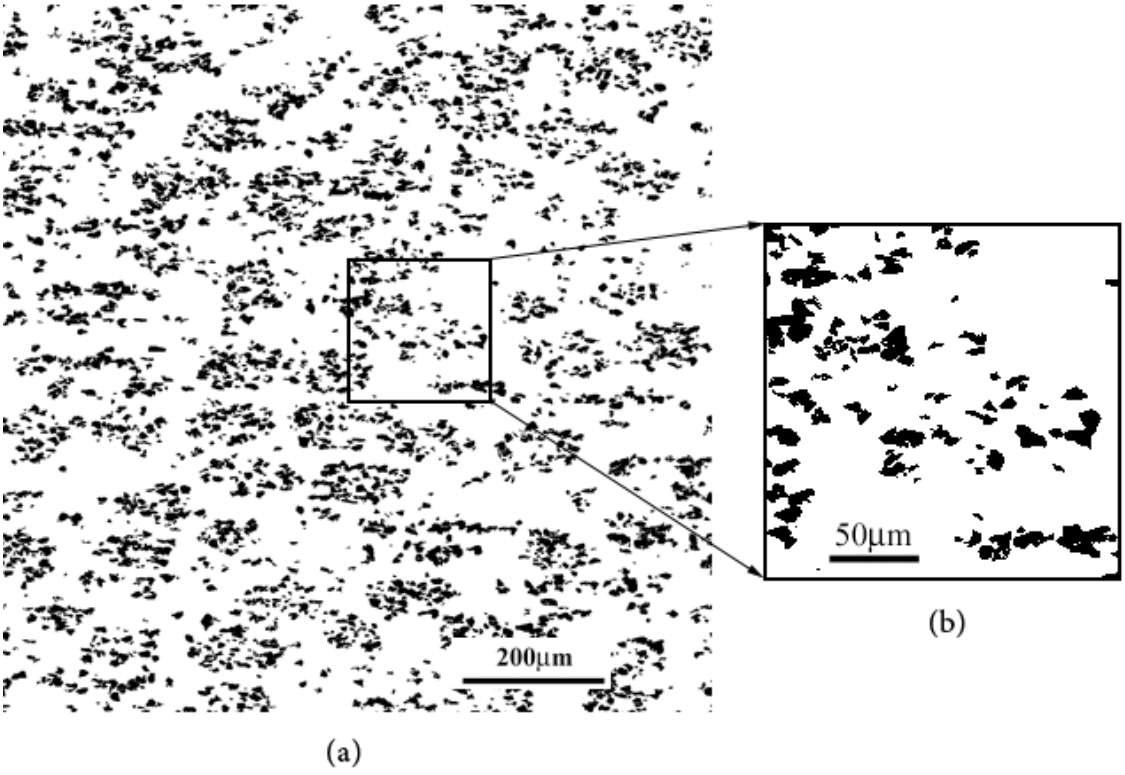


Figure 4.43: (a) Montage of simulated 3D microstructure with low volume fraction (15%) of SiC particles and PSR=8.1 (b) Magnified view of the outlined region in (a)

Finite element based simulations can be carried out on the virtual and real 3D microstructure images for *realistic* parametric studies on the variations in the micromechanical response. The resulting data can provide useful information for materials by design, and the methodology can cut down on the number of experiments (and therefore, time and resources required) for developing new composites and for optimizing the properties of the existing composites. The implementation of the 3D microstructural images in the FE based simulations of the micro-mechanical behavior of the PSR 2.0 DRA composite which has been carried out by Dr. Arun Sreeranganathan [110] as a part of his doctoral thesis research is presented in the next section.

4.2.7 Implementation of Realistic Simulated 3D Microstructures in FE-Based Simulations of Mechanical Behavior

4.2.7.1 Micromechanical Analysis of Real and Simulated 3D Microstructures

The real and simulated 3D microstructural volumes of PSR 2.0 DRA composite were meshed with second-order tetrahedral elements (Abaqus C3D10M elements) using Simpleware, a commercial software developed at University of Exeter, UK for the conversion of 3D images into high quality meshes [111]. Figure 4.44 shows the FE mesh created from the real PSR 2.0 DRA composite microstructure of volume 200 μm × 200 μm × 100 μm. The mesh size is on the order of 3 μm, and it can be seen from Figure 4.44 that the mesh is quite refined and conforms well to the underlying 3D microstructure. The mesh contained over 5 million degrees of freedom.

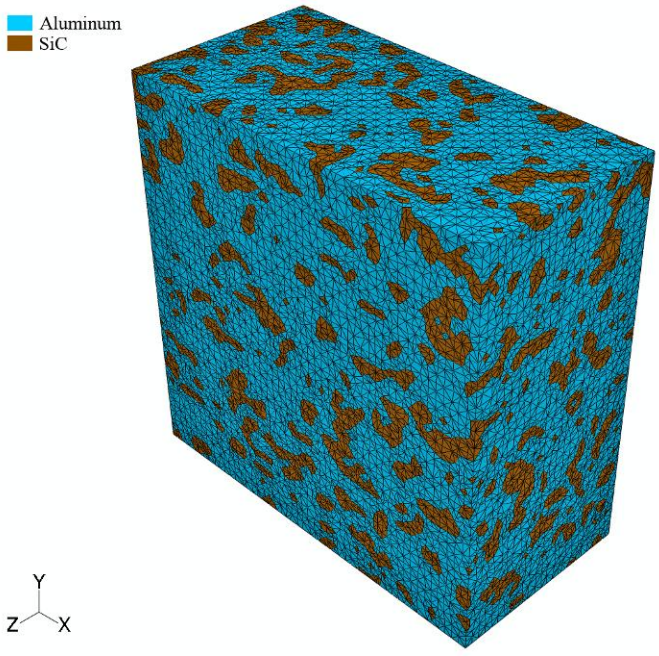


Figure 4.44: FE-mesh for real reconstructed 3D microstructure segment of the PSR 2.0 DRA composite [110]

The FE mesh generated using Simpleware was imported into the commercial finite element package, Abaqus, for subsequent micromechanical analysis. All the FE analyses in this research were performed quasi-statically using Abaqus/Explicit (6.7-1) and were run at the University of Illinois' National Center for Supercomputing Applications (NCSA) supported by the National Science Foundation through TeraGrid resources [112]. Figure 4.45 shows the computed stress-strain curves for the real and simulated DRA microstructures under uniaxial loading in the extrusion direction along with the experimentally determined stress-strain curve for the composite. The experimental curve is shown as a range accounting for the variability observed in the tensile test results. All the three curves are in good agreement with each other. The damage initiation and growth in the composite is controlled by the distribution of local maximum principal stress (σ_1) in the SiC particles and the plastic strain distribution in the matrix. Therefore, it is of interest to compare these local stress/strain distributions in the real and simulated microstructures. Figure 4.46 shows the complementary cumulative distribution of the maximum principal stress within the SiC particles in the models, where Y-axis is the fraction of the SiC integration points with maximum principal stress higher than a given value. Similarly, Figure 4.47 gives the complementary cumulative distribution of equivalent plastic strain in the matrix. Figures 4.46 and 4.47 show that the stress/strain distributions in the real and simulated microstructures are in good agreement with each other, demonstrating that the simulated microstructure mimics the micromechanical response of the real microstructure.

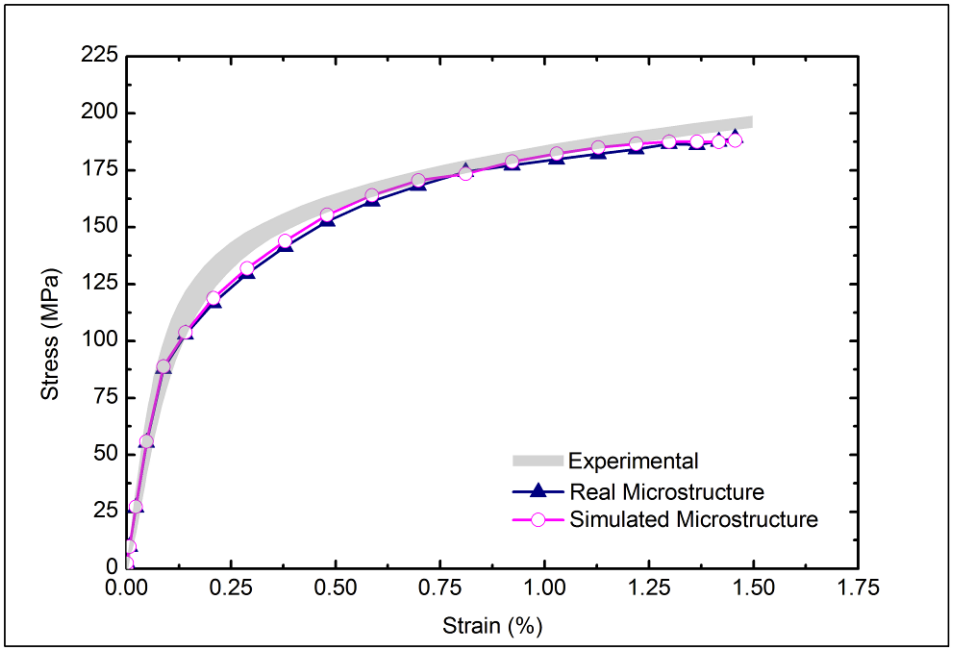


Figure 4.45: Computed stress-strain curves of the 3D microstructural images of the real and simulated PSR 2.0 DRA composite along with the *experimentally measured* stress-strain curve of the composite [110].

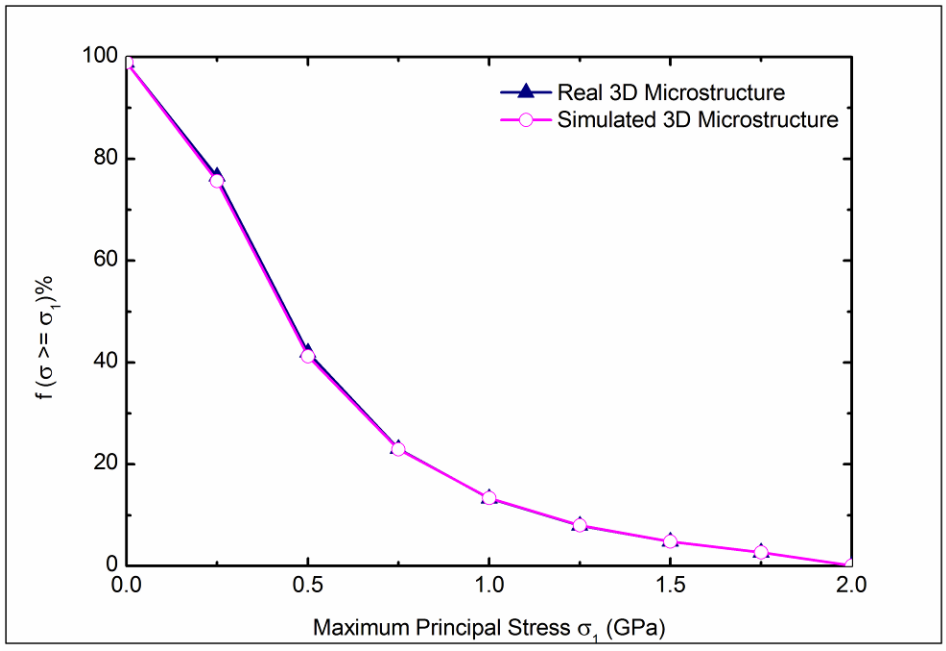


Figure 4.46: Complementary cumulative distribution of maximum principal stress in SiC particles for real and simulated PSR 2.0 DRA microstructural volumes. Y-axis is the fraction of the SiC integration points with maximum principal stress higher than a given value [110].

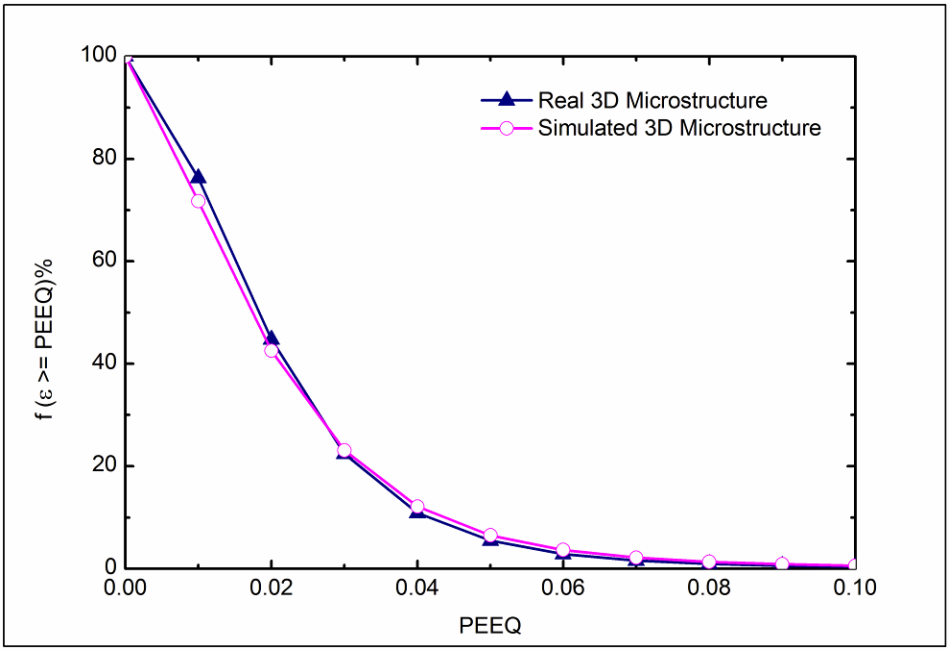


Figure 4.47: Complementary cumulative distribution of equivalent plastic strain in the matrix for real and simulated PSR 2.0 DRA microstructural volumes [110].

4.2.7.2 Micromechanical Analysis of Virtual Microstructures

Five different 2.0 PSR microstructures were simulated with varying volume fractions of SiC particles, ranging from 10% to 30% at increments of 5%. These simulated microstructures have the same spatial clustering, anisotropy, and size/shape distribution of the SiC particles and they differ only in the SiC volume fraction. Figure 4.48 shows one such simulation, where the spatial clustering of the particles is statistically similar to that in Figures 4.8 and 4.14 but the volume fraction of SiC particles (15%) is lower than that in the Figures 4.8 and 4.14 (28%). The simulated volumes were of size 200 μm × 200 μm × 200 μm. The ability to simulate any required volume size is another advantage of realistic simulation models. For heterogeneous microstructures with long-range spatial correlations, one of the current limitations in reconstructing the 3D

microstructures is the effort involved in the reconstruction of large volumes (for example, the number of polishing cycles involved in a serial-sectioning procedure). Once a simulation model is validated, it can be used to create microstructural volumes of any size that may be required to be considered a microstructural representative volume element.

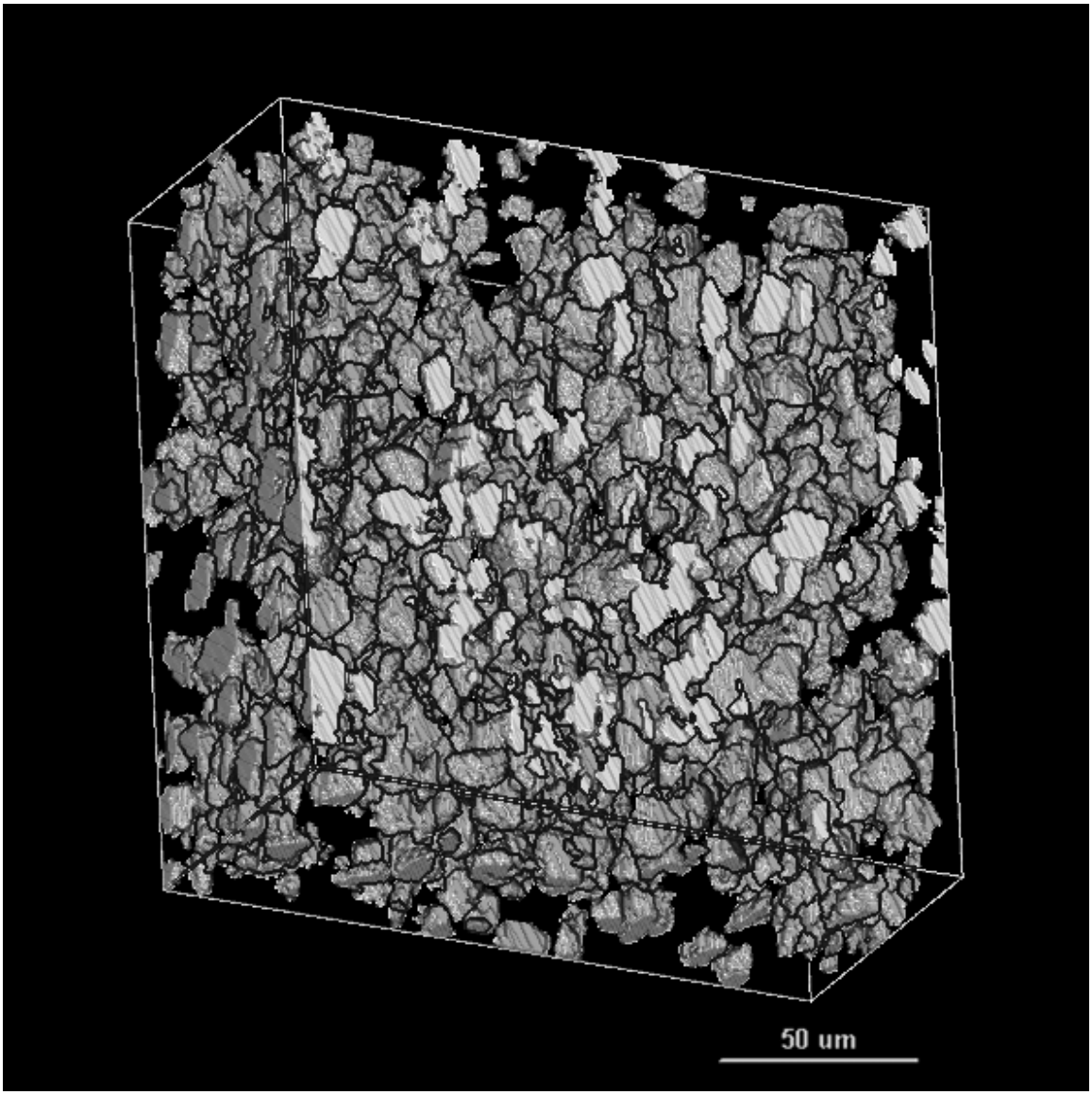


Figure 4.48: Small segment of simulated 3D microstructure with low volume fraction (15%) of SiC particles and PSR=2.0

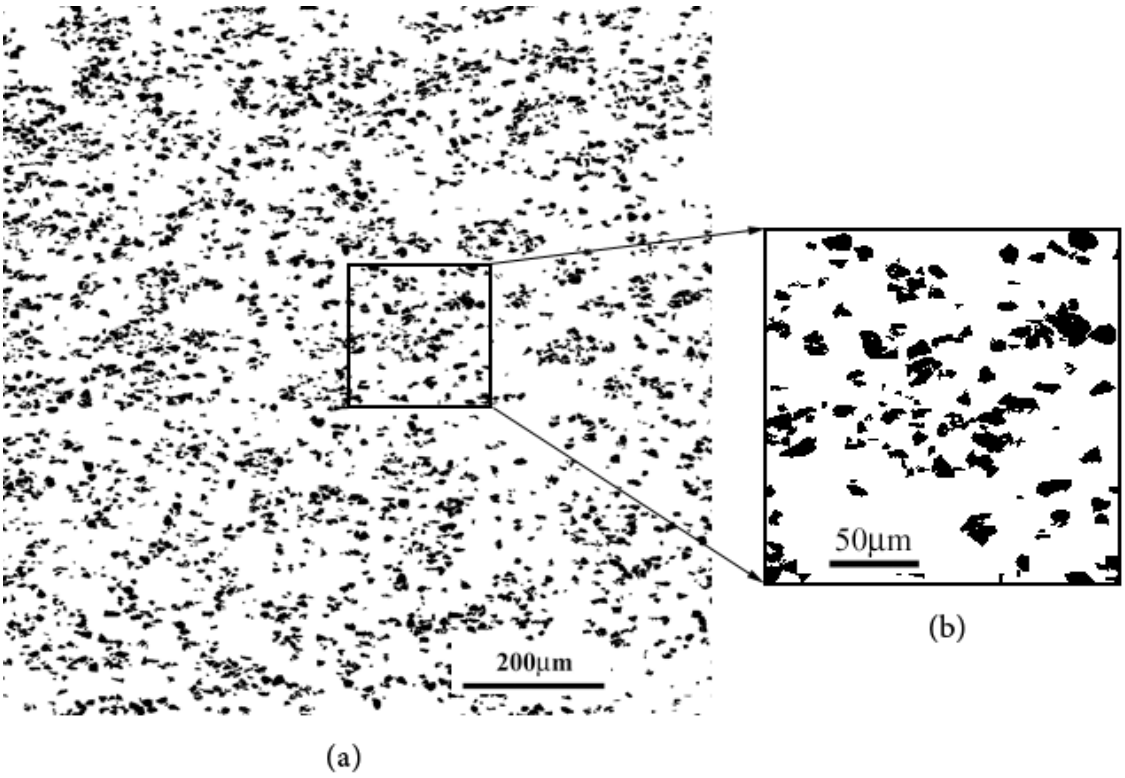


Figure 4.49: (a) Montage of simulated 3D microstructure with low volume fraction (15%) of SiC particles and PSR=2.0 (b) Magnified view of the outlined region in (a)

All the five simulated microstructural volumes were incorporated in finite element models to simulate the micromechanical behavior of these composites. Figure 4.50 shows the FE mesh for the five different volumes.

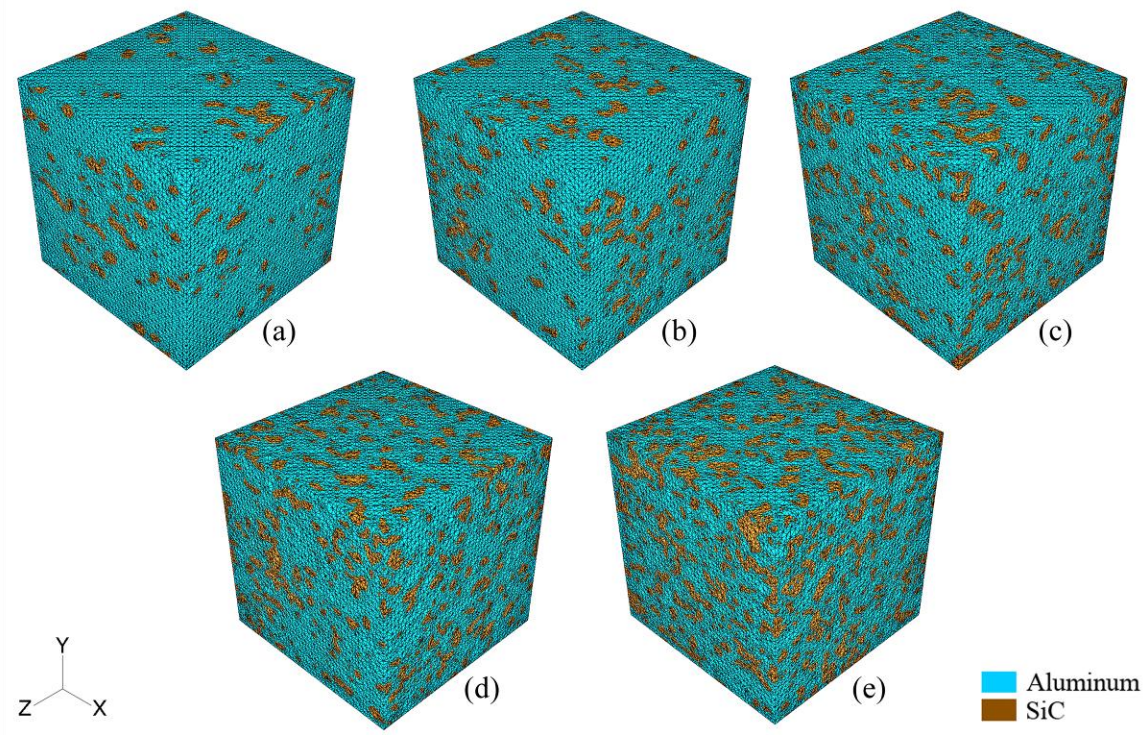


Figure 4.50: FE mesh for simulated microstructural volumes containing (a) 10% (b) 15% (c) 20% (d) 25% and (e) 30% SiC particles [110].

The computed stress-strain curves for uniaxial loading along the extrusion direction for all the models are illustrated in Figure 4.51. The Young's modulus and 0.2% yield strength values for the five models are given in Table 4.3. As expected, the modulus and yield strength values of the composite increase with an increase in SiC particulate volume fraction. Higher matrix strain-hardening is observed initially for composites with higher SiC volume fraction because of localized high strain regions but the strain-hardening rate approaches that of the unreinforced matrix when strains are in the regime of fully developed plastic flow.

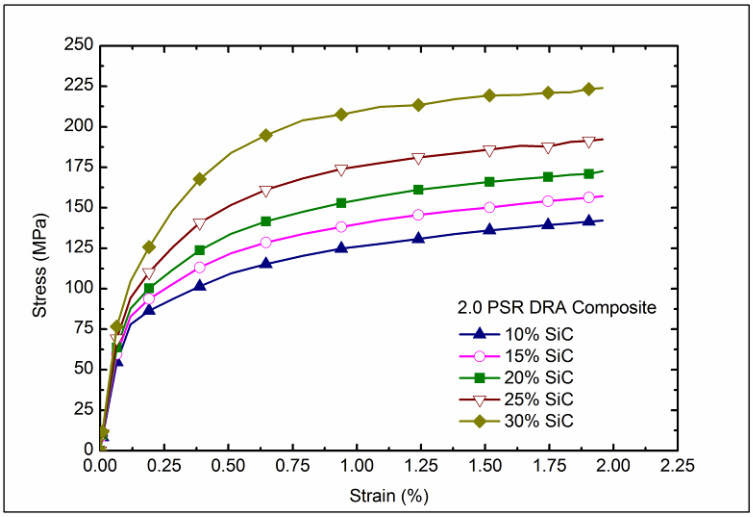


Figure 4.51: Computed stress-strain curves for the simulated virtual microstructural volumes shown in Figure 4.50 [110].

Table 4.3: Computed modulus and yield strength values from 3D FE simulations of virtual microstructures with varying SiC volume fractions

Volume percentage of SiC	Elastic Modulus (GPa)	Yield Strength (Mpa)
10	87	90
15	95	98
20	102	107
25	111	118
30	124	140

4.3 Computer Simulations of Realistic 3D Microstructures of Boron Modified Ti-6Al-4V Composites

As mentioned earlier, the orientation of each individual SiC particle was not changed for the simulation of the 3D microstructures of the DRA composites since the morphologies of the particles are isotropic. Nevertheless, the simulation methodology is capable of allowing controlled rotation of the particles/whiskers. This feature is presented through its application to the 3D microstructure of discontinuously reinforced Ti-TiB composites that have been compacted, or compacted and subsequently extruded. The next sub-section describes the experimental part of the work including the 3D microstructure reconstruction. The microstructure simulation technique and its application to Ti-TiB composite microstructures are described in the subsequent sub-sections.

4.3.1 Material

Addition of small amounts of boron during high temperature processing of titanium alloys such as Ti-6Al-4V (Ti64) has emerged as an effective way to improve the strength and stiffness of these alloys, while at the same time maintaining their densities and fracture properties. The properties improvements are attributed to the *in situ* formation of short TiB whiskers during high temperature processing of these alloys [113], The phase diagram of the Ti-B binary system (see Figure 4.52) shows the formation of the TiB intermetallic phase due to a eutectic reaction [114-115], with eutectic point at B = 1.64 wt.% for Ti-B binary system and B = 1.55 wt.% for Ti-6Al-4V-B (Ti64-B) quaternary system [113]. Boron is completely soluble in liquid titanium but is essentially insoluble in the solid titanium phases (high temperature β as well as room temperature α phase). The density of TiB is comparable to that of titanium but the

stiffness is about 3-4 times that of conventional titanium. The phase diagram reveals that for hypereutectic Ti-B alloys (B > 1.55 wt.% for Ti64-B system), along with the eutectic phase, TiB also forms as a coarse primary phase directly precipitating from the liquid phase. On the other hand, only eutectic TiB whiskers are present in hypoeutectic alloys (B ≤ 1.55 wt.% for Ti64-B system). The hypoeutectic alloys are generally classified as boron-modified titanium alloys, whereas the hypereutectic alloys are referred to as Ti-TiB composites. These titanium alloys can be made using a variety of techniques including conventional casting and powder metallurgy processes, and can also be subjected to conventional thermomechanical processing operations such as forging, extrusion, and rolling to produce desired shapes with tailored isotropic or anisotropic microstructures (e.g. isotropic or anisotropic orientation of the TiB whiskers). The wide range of compositions and processing methods available for Ti-B materials requires a thorough understanding of the relationships between the processing, microstructure and properties. Information from 2D microstructural characterization is not fully adequate for this purpose due to the complexity of the microstructures of these materials. Therefore, it is of interest to develop simulation methodologies that incorporate realistic 3D complex shapes/morphologies and morphological orientation distribution of the Ti-B whiskers and/or coarse primary Ti-B particles in the 3D microstructure models and simulations, which can be subsequently implemented in the computational models and simulations of the mechanical behavior and processing of the Ti-B materials.

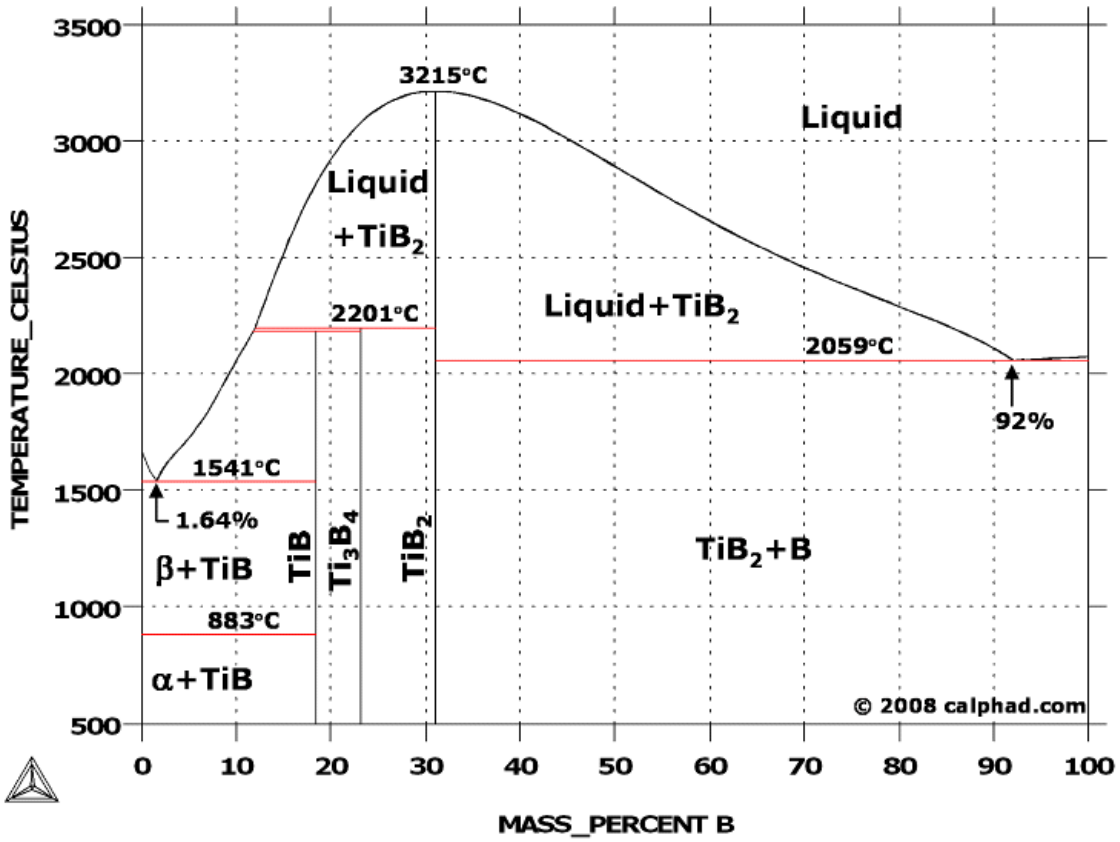


Figure 4.52: Ti-B phase diagram [114]

Ti-6Al-4V is the most widely used titanium alloy, accounting for more than half of the overall worldwide titanium tonnage [116]. The aluminum addition stabilizes the hcp α phase, which increases the overall strength of the alloy. The vanadium addition introduces bcc β phase in the α matrix, increasing ductility and fracture toughness. Ti-6Al-4V is considered an α-rich α+β alloy. In boron modified Ti-6Al-4V alloys, the microstructure consists of *in situ* formed TiB whiskers in a matrix of equiaxed α grains and retained β phase decorating the grain boundaries. It has been shown that the TiB whiskers effectively pin the grain boundaries of the titanium alloy matrix so that a fine grain structure is retained even well above the β-transus and after cooling back into the

α+β phase field [117-118]. Boron-modified titanium alloys have the potential to expand the usage of titanium and are attractive for a variety of applications in the automotive, aerospace, biomedical, and sporting goods industries.

In this study, experiments have been performed on Ti-6Al-4V-1.6B composites (all compositions are in weight percent) produced via a pre-alloyed powder metallurgy approach at Crucible Research Corporation, Pittsburgh, Pennsylvania. In this process, the liquid melt of Ti64 containing boron is rapidly solidified using inert gas atomization to produce Ti64-1.6B powder. The powder metallurgy and extrusion processes were performed by Dr. S. Tamirisakandala at Wright-Patterson Air Force Laboratory. The pre-alloyed powder is blind die compacted to produce the compact Ti64-1.6B composites. The compacted composites are subsequently extruded at 1100 ˚C with an extrusion ratio of 16.5:1 to produce the extruded Ti64-1.6B composites. The processing details are reported elsewhere [118]. The selected composition lies in the hypereutectic regime. Therefore, the microstructures contain both fine eutectic TiB whiskers and coarse primary TiB particles. Considering the processing conditions chosen in these experiments, it is reasonable to expect that in the extruded composites, the TiB whiskers would have anisotropic orientations with majority of the whiskers aligned along the extrusion direction, and in the compacted composites without the subsequent extrusion, the TiB whiskers would have uniform random spatial orientations leading to uniform isotropic microstructure.

4.3.2 3D Microstructure Reconstruction

In the present work, the same montage-based serial sectioning technique reported in the previous section [107, 119-120] is used to reconstruct the 3D microstructures of

Ti64-1.6B composites, where each 2D layer is a montage of many contiguous fields of view (FOV) stitched together digitally, and a 3D volume is created by aligning these layers and interpolating the thickness between them. The experimental metallography and 3D microstructure reconstruction were performed by Dr. Scott Lieberman [119, 121] as a part of his doctoral thesis research. Each montage serial section has a size of 3500×3500 pixels with a pixel size of 0.2 μm. Figure 4.53 shows a typical montage serial section image of the compacted and extruded Ti64-1.6B composite microstructure. The extrusion direction is normal to the serial section planes. Both primary and eutectic TiB phase are present in this image. In this 2D serial section plane, the TiB whiskers appear to have random orientation, while in 3D, the TiB whiskers are aligned along the extrusion direction. Therefore, it is necessary to utilize the 3D reconstruction to reveal the true microstructure of the material. Figure 4.54 depicts a corresponding montage serial section image for the compacted Ti64-1.6B composite that was not subsequently extruded. Once the montage of the first serial section is created and stored in the computer. A small thickness of the specimen is removed by polishing (averaged 0.7 in the present case), and then a second montage is created at the region exactly below that in the first metallographic plane. This polish-montage-polish procedure was repeated to obtain a stack of 75 montage serial sections. Figure 4.55 depicts a stack of 20 aligned montage serial sections for compacted and extruded Ti64-1.6B composite microstructure. For computer simulations of realistic 3D microstructures, it is advantageous to have cubic voxels (3D analog of pixels) in which the resolution along the X, Y, and Z directions is the same. To achieve this, each experimental montage serial section image was re-sized to the size of 1000×1000 pixels at the resolution of 0.7 μm per pixel. This leads to a

reconstructed 3D microstructure volume of 1000 × 1000 × 75 voxels with a voxel size of 0.7 × 0.7 × 0.7 μm.

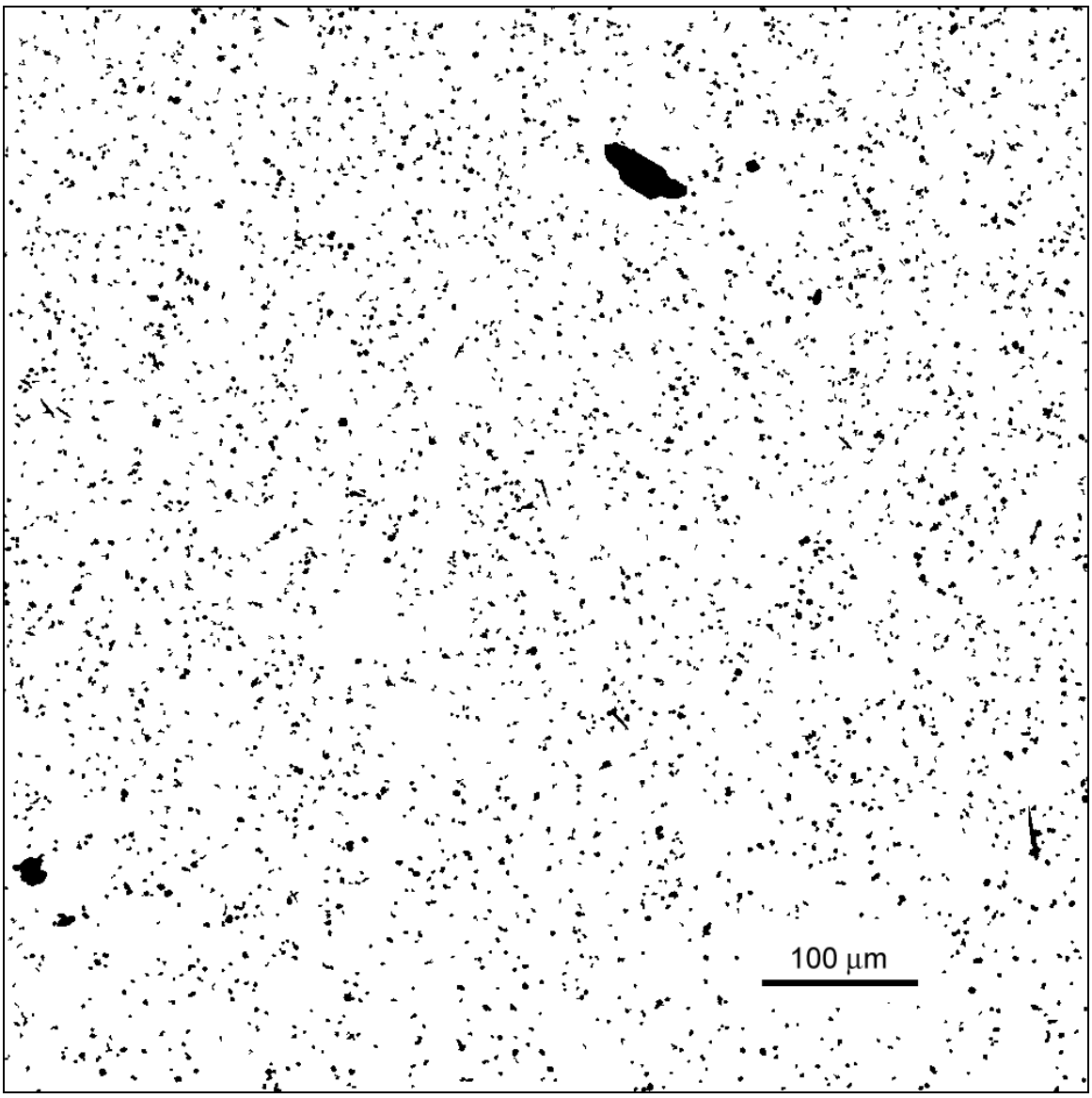


Figure 4.53: Montage serial section image of the compacted and extruded Ti64-1.6B composite microstructure

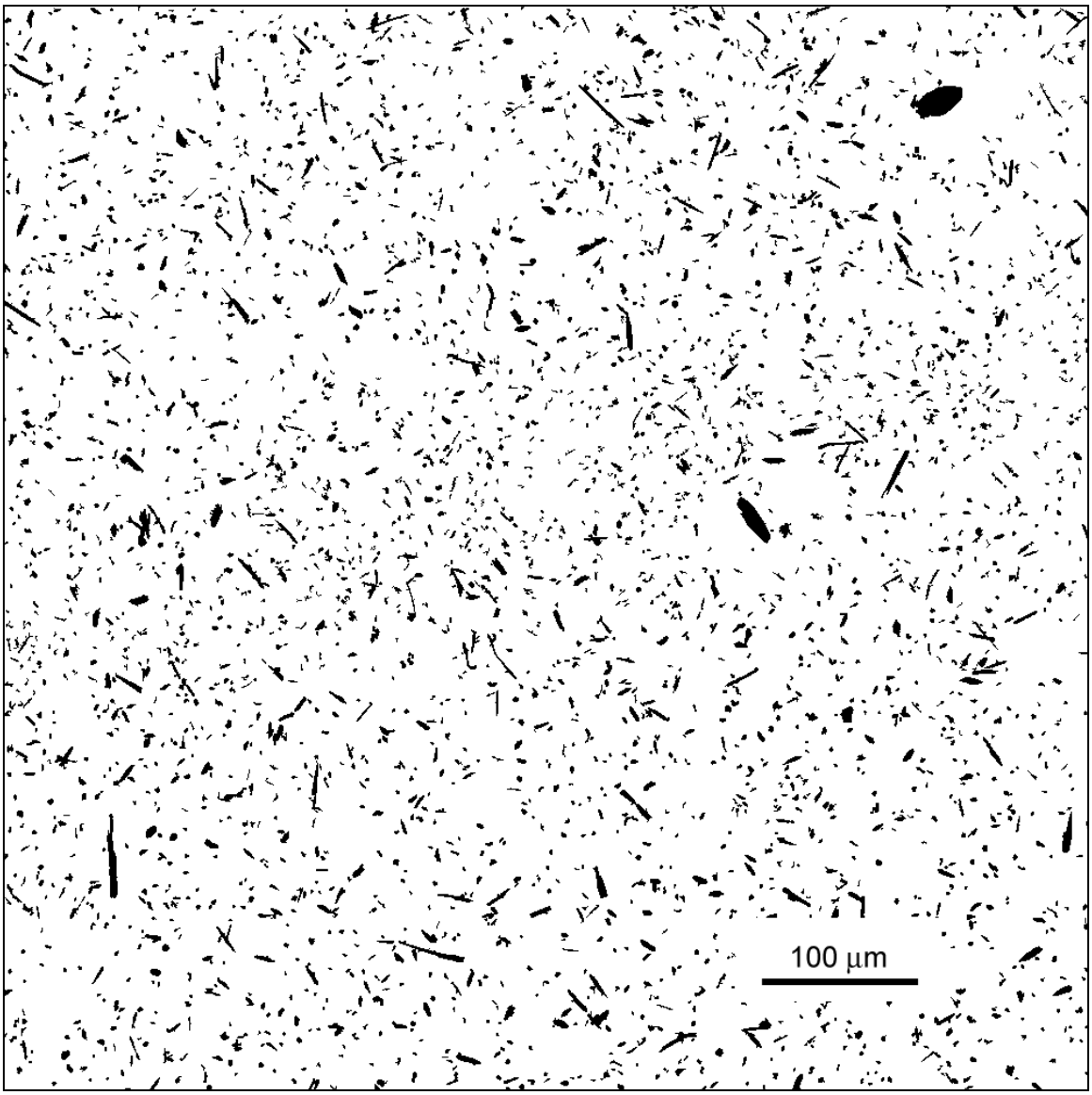


Figure 4.54: Montage serial section image of the compacted (but not extruded) Ti64-1.6B composite microstructure

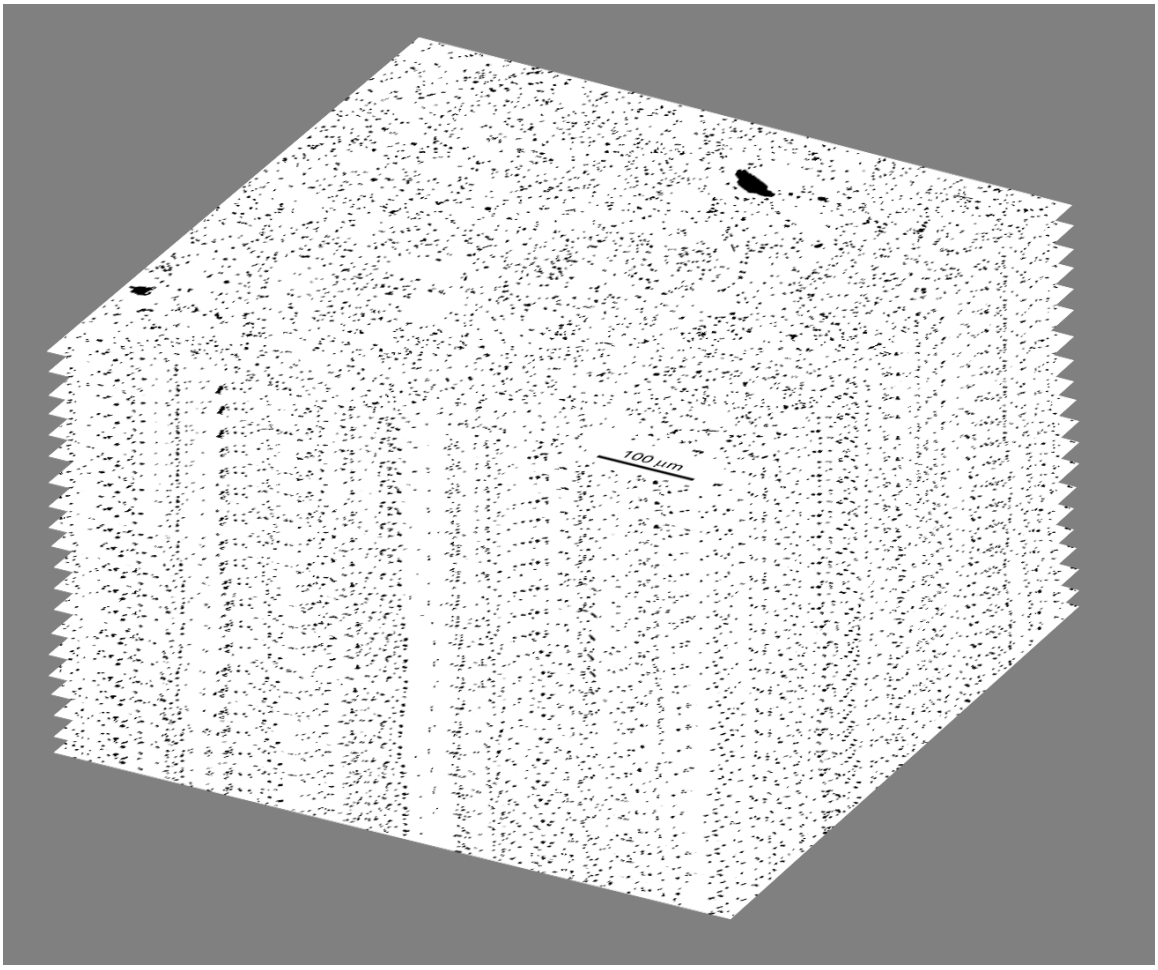


Figure 4.55: A stack of 20 aligned montage serial sections for compacted and extruded Ti64-1.6B composite microstructure

Figure 4.56 shows a small segment of compacted and extruded Ti64-1.6B composite's 3D microstructure reconstructed from the serial section images using surface rendering. In this reconstructed 3D microstructure segment, most of the TiB whiskers are aligned parallel to the extrusion direction (which is normal to the top and bottom faces of the segment). Figure 4.57 shows a small segment of reconstructed 3D microstructure of the compacted (but not extruded) Ti64-1.6B composite. Note that the TiB whiskers have uniform random morphological orientations in this 3D microstructure. Although the

morphological orientation distribution of TiB whiskers is different in the compacted and extruded microstructure as compared to compacted but not extruded microstructure, the spatial arrangement of TiB whiskers appears to be uniform random in both materials. Therefore, although the extrusion process leads to TiB whisker rotations, it does not give rise to spatial clustering.

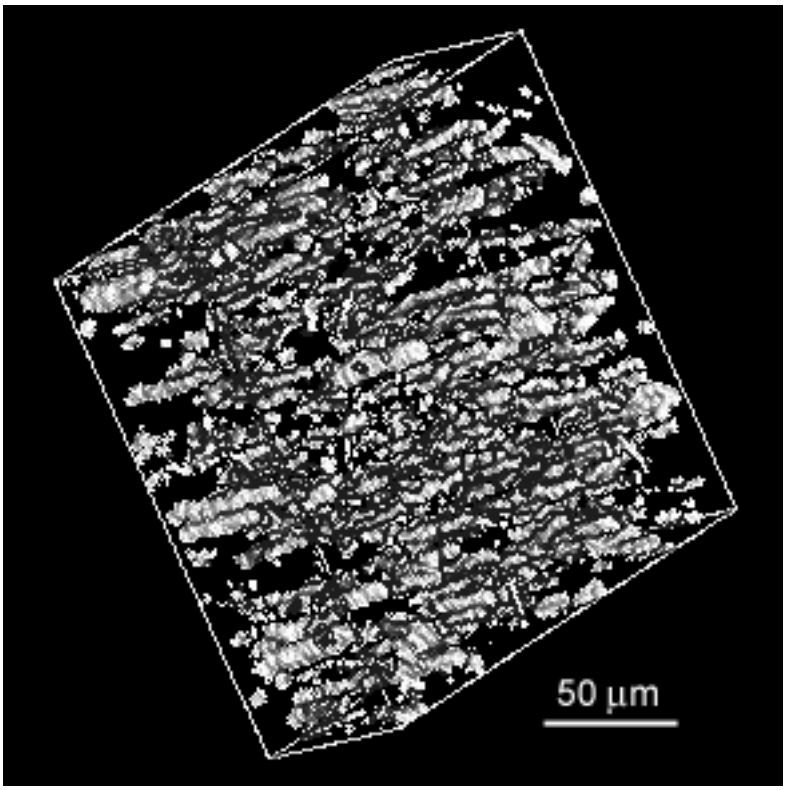


Figure 4.56: Small segment of 3D microstructure of compacted and extruded Ti64-1.6B composite

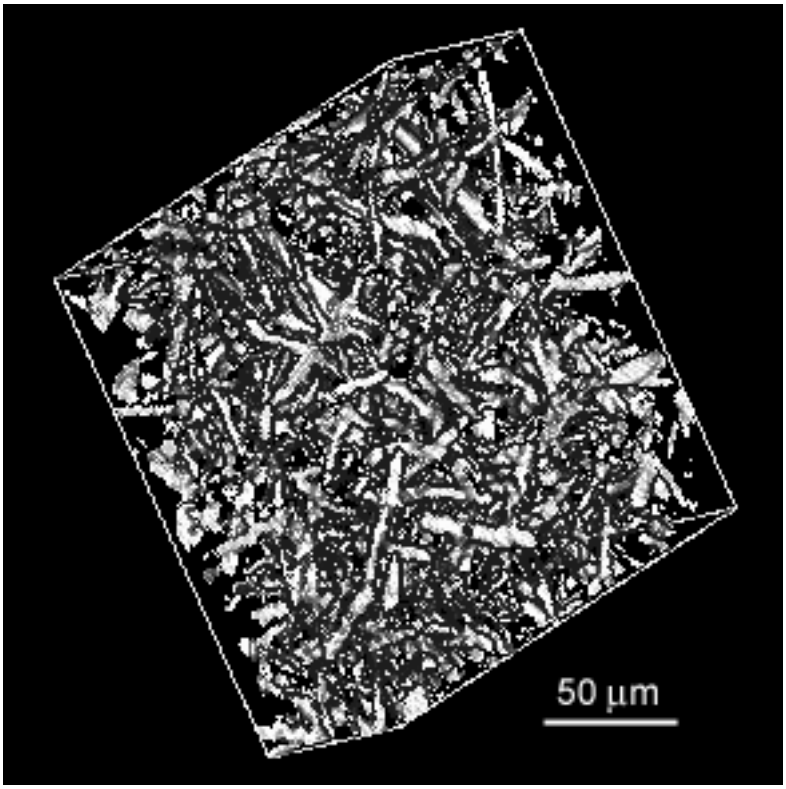


Figure 4.57: Small segment of 3D microstructure of compacted (but not extruded) Ti64-1.6B composite

4.3.3 Quantitative Microstructure Characterization and Representation

As described in the previous section, experimentally measured two-point correlation functions in three perpendicular directions are used to represent the 3D microstructure. Figure 4.58 compares two-point correlation functions for the compacted and extruded Ti64-1.6B composite measured along the extrusion direction, long transverse direction and short transverse direction. The correlation function approaches 1.0 at the longer distance along the extrusion direction than along the transverse direction, which is expected since the TiB whiskers are aligned in the extrusion direction. On the other hand, the compacted (but not extruded) Ti64-1.6B composite has an isotropic

morphological orientation distribution and spatial arrangement of TiB whiskers. This is confirmed by comparing the two-point correlation functions for the compacted Ti64-1.6B composite measured along three perpendicular directions, namely, X, Y and Z direction, as illustrated in Figure 4.59.

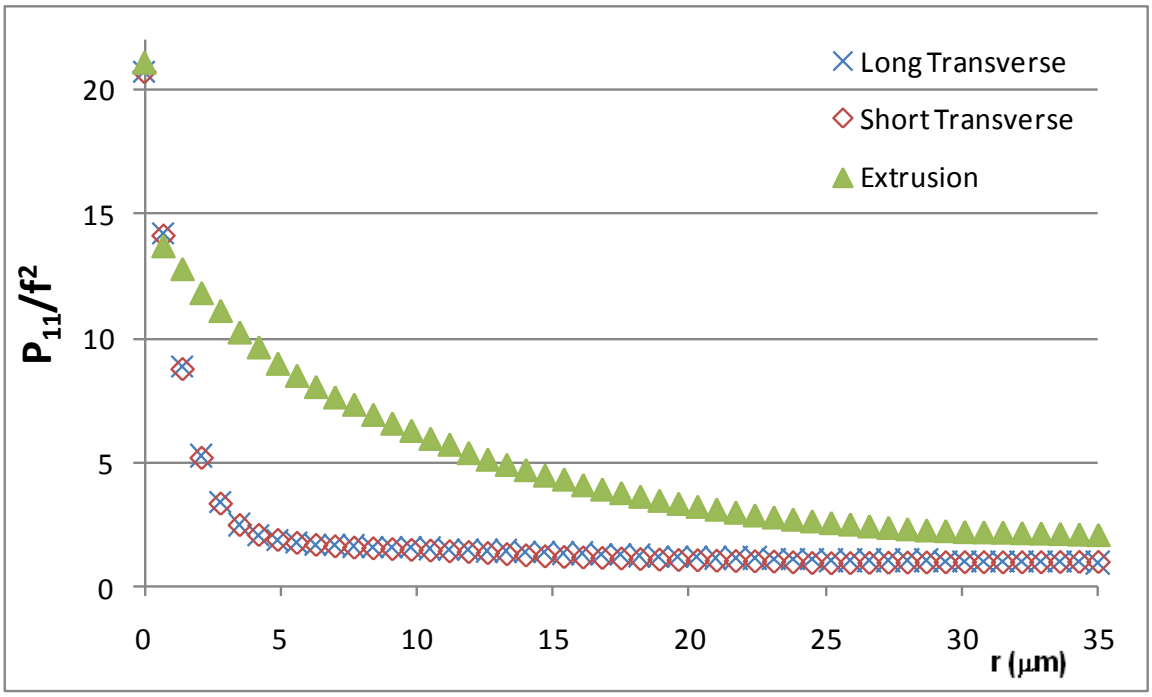


Figure 4.58: Two-point correlation functions for the compacted and extruded Ti64-1.6B composite measured along the extrusion direction, long transverse direction and short transverse direction.

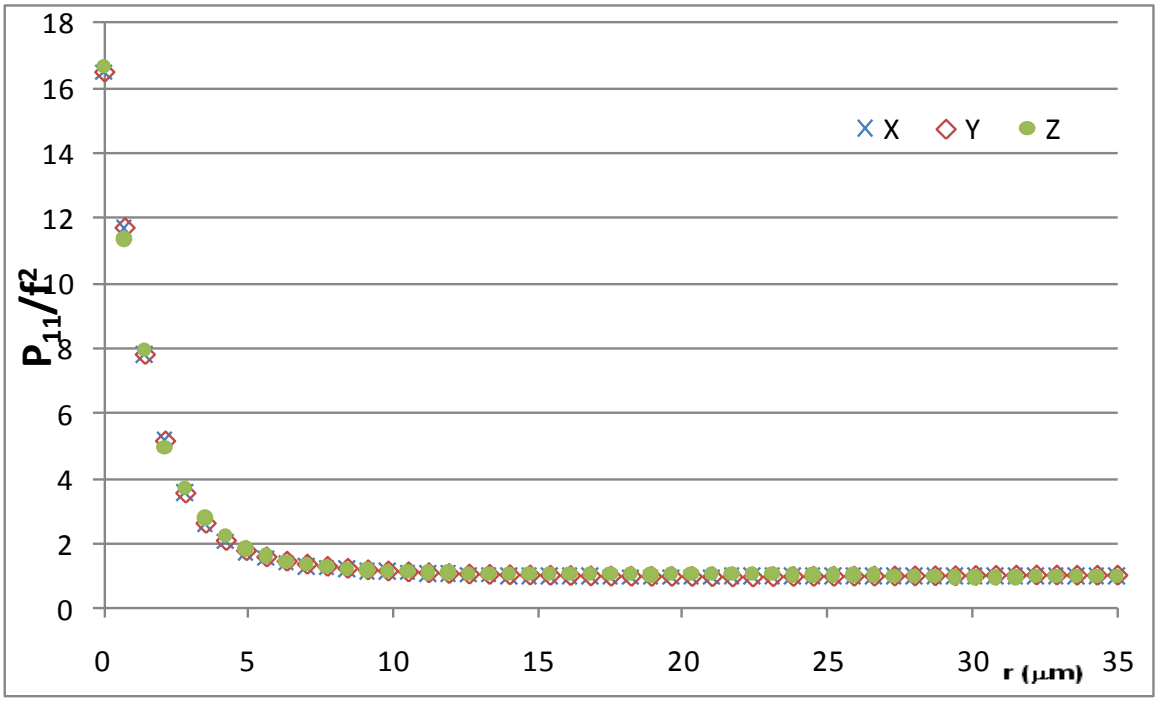


Figure 4.59: Two-point correlation functions for the compacted (but not extruded) Ti64-1.6B composite measured along three perpendicular directions, namely, X, Y and Z direction.

The methodology for realistic simulations of the compacted and extruded Ti64-1.6B composite microstructure having both fine TiB whiskers and coarse primary TiB particles is presented in the next section.

4.3.4 Computer Simulation of 3D Microstructures

To simulate 3D microstructure of the Ti64-TiB composite having realistic complex particle shapes/morphologies and isotropic/anisotropic orientations, 1500 TiB whiskers/particles including a few primary TiB particles were extracted from the extruded Ti-TiB composite to represent the size and shape distribution of the TiB whisker/particle population in the present composites.

Next, the individual TiB whisker/particle is placed at uniform random locations until the desired overall volume fraction and size/shape distribution of particles is achieved. Note that in the extruded composite microstructure simulation, the particle orientations are kept the same as those in the corresponding real extruded microstructure. However, to simulate microstructures having different degrees of morphological anisotropy that can arise due to variations in the amount and/or temperature of deformation processing, the whiskers/particles need to be rotated by a specified amount before they can be placed in the simulation space. A rotation can be performed either by a forward mapping or by an inverse mapping. In the forward mapping, input voxel is first identified in the origininal volume. Its coordinates (X_n, Y_n, Z_n) are then mapped to a new coordinates (X'_n, Y'_n, Z'_n) in the output volume using rotation transformation matrix (R) as follows:

$$\begin{bmatrix} X'_n \\ Y'_n \\ Z'_n \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \begin{bmatrix} X_n \\ Y_n \\ Z_n \end{bmatrix} \quad \text{Eq. 6}$$

In the discrete domain, a forward mapping may cause two types of artifacts: holes and overlaps. Because of this disadvantage, inverse mapping method is generally used for digital image rotation. In the inverse mapping, integer voxel locations in the output volume are mapped back to fractional voxel locations in input volume and fetch the approximated input voxel value at the nearest integral position in the input volume as follows:

$$\begin{bmatrix} X_n \\ Y_n \\ Z_n \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix}^{-1} \begin{bmatrix} X'_n \\ Y'_n \\ Z'_n \end{bmatrix} \quad \text{Eq. 7}$$

Consider simulation of compacted composite microstructure with isotropic random morphological orientation distribution using aligned TiB whiskers extracted from the extruded composite microstructure. The extracted whiskers are aligned in the extrusion direction. Therefore, each whisker needs first to be uniform randomly rotated in the 3D space. Generating a uniform random rotation in 3D-space is not a trivial exercise. Simply using evenly distributed sets of the three rotational angles leads to a non-uniform sampling since the polar areas are heavily oversampled. In the present work, the random rotation matrix (R) is generated using the method described by Arvo [122]. The technique involves rotation of objects located at the north pole vertically by a random amount, and then rotation the axis of the north pole to a random position on the sphere. The rotation matrix is constructed as follows [122]:

$$R = (2VV^T - I)\begin{bmatrix} \cos(2\pi x_1) & \sin(2\pi x_1) & 0 \\ -\sin(2\pi x_1) & \cos(2\pi x_1) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{Eq. 8}$$

Where $V = \begin{bmatrix} \cos(2\pi x_2)\sqrt{x_3} \\ \sin(2\pi x_2)\sqrt{x_3} \\ \sqrt{1-x_3} \end{bmatrix}$ and x_1, x_2 and x_3 are three independent uniform random variables between 0 and 1.

Once desired overall volume fraction, orientation and size/shape distribution of particles are achieved, the two-point correlation functions of the TiB particles in the

simulated microstructure are computed and compared with the experimentally measured two-point correlation functions. Simulation parameters are varied until a satisfactory match between the two-point correlation functions of the real and simulated microstructures is achieved. The simulation parameters that can be changed to vary the microstructure are as follows.

- Volume fraction of TiB whiskers/particles,
- TiB whisker/particle size distributions,
- TiB whisker/particle orientation distributions.

The computer code for the 3D microstructure simulations is given in Appendix B.8.

4.3.5 Computer Simulation Results

The realistic 3D microstructure simulation technique has been applied to simulate the 3D microstructures of extruded and compacted Ti64-1.6B composites containing fine eutectic TiB whiskers and coarse primary TiB particles. An input volume of 1000 × 1000 × 75 voxels of the real extruded composite microstructure was used to extract the real particle morphologies. The simulation code was applied on these input whiskers and particles to generate two different simulated 3D microstructural volumes having a volume of 1000 × 1000 × 100 voxels with voxel size of 0.7 × 0.7 × 0.7 μm and each containing over ten thousand particles. Figure 4.60 depicts a small segment of microstructural volume of extruded Ti64-1.6B composites simulated without particle rotation, which is comparable to the real 3D microstructure shown in Figure 4.56. Figure 4.61 depicts a small segment of microstructural volume of extruded Ti64-1.6B composites simulated with the same set of input particles but rotated using uniform

random rotation matrix constructed by Eq. 8. The simulated microstructure is comparable to the real 3D microstructure shown in Figure 4.57. Each simulated volume can be divided into 100 simulated montage serial sections having a size of 1000 × 1000 pixels at a resolution of 0.7 μm per pixel (see Figures 4.62 and 4.63).

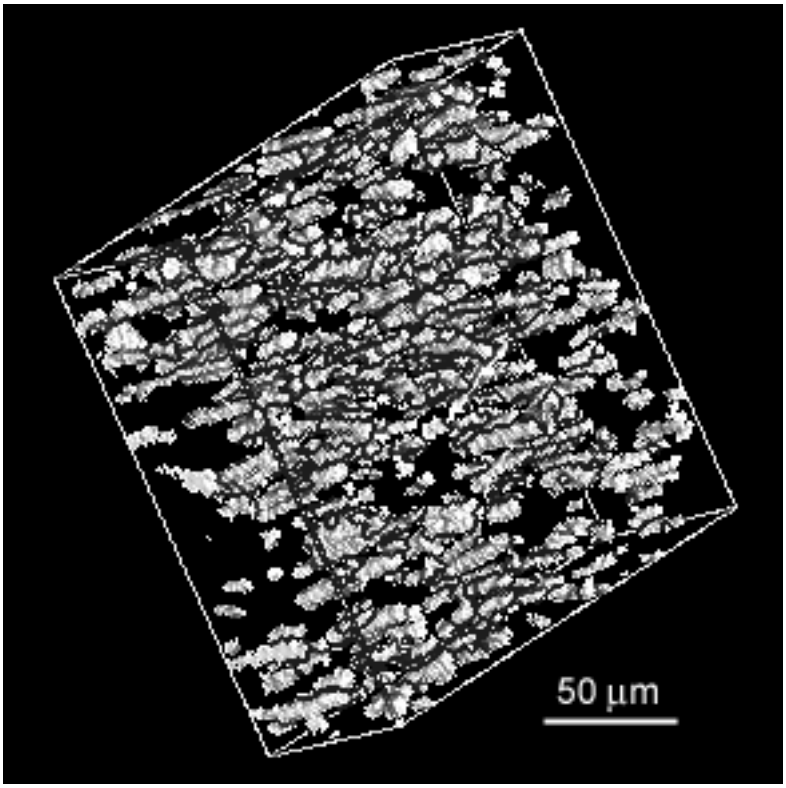


Figure 4.60: Small segment of simulated 3D microstructure of extruded Ti64-1.6B composite

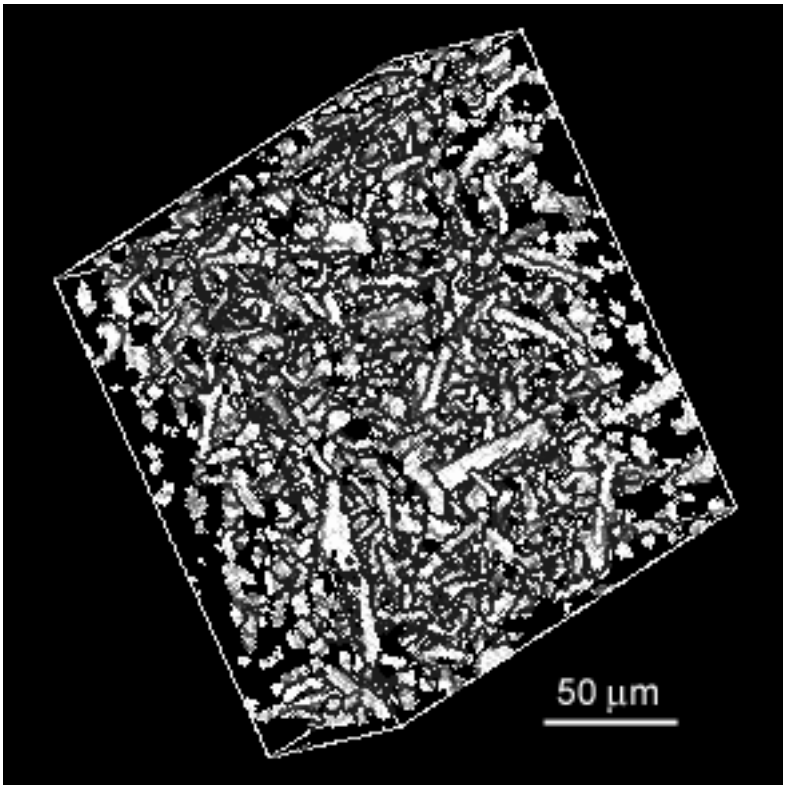


Figure 4.61: Small segment of simulated 3D microstructure of compacted Ti64-1.6B composite

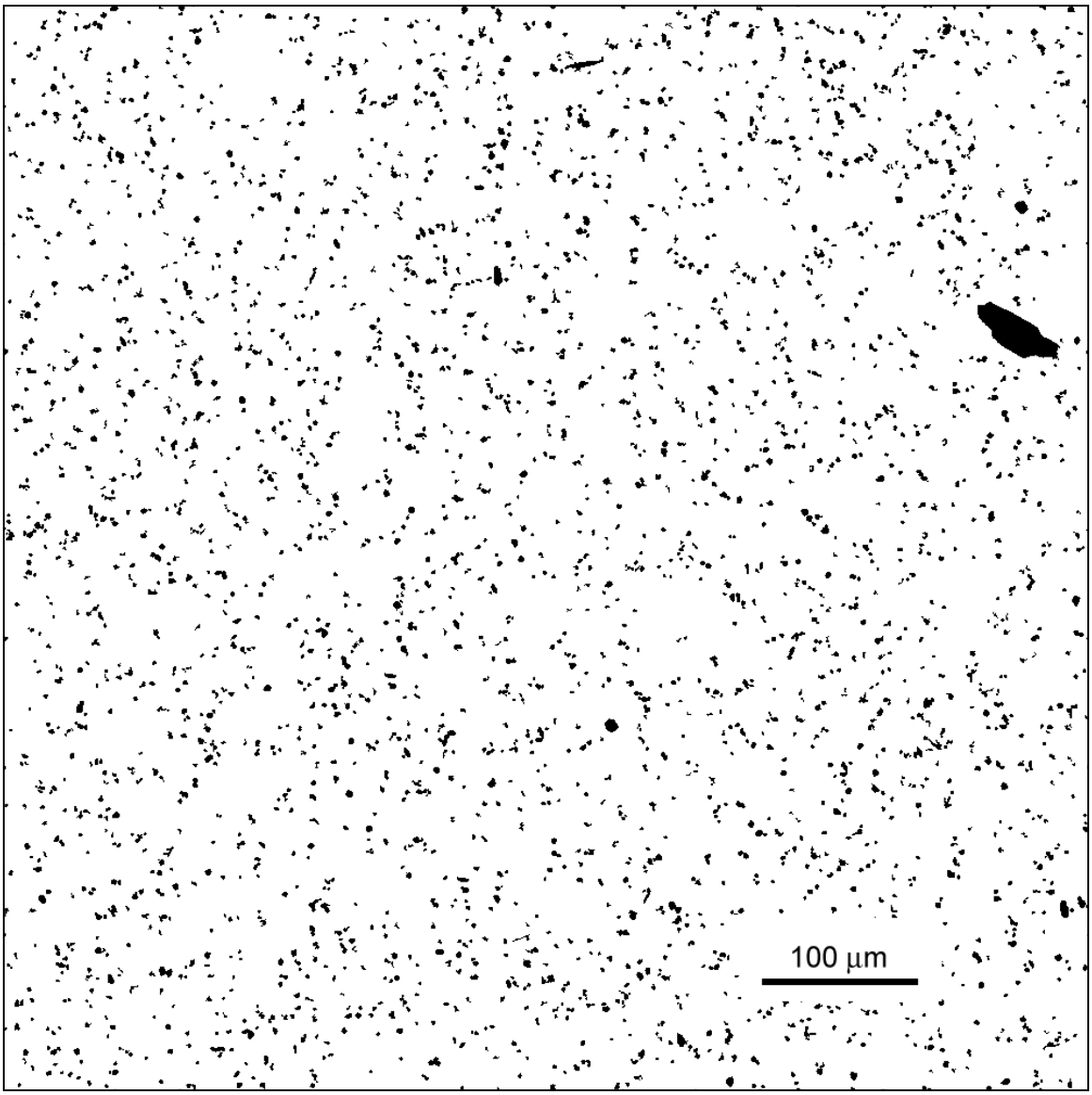


Figure 4.62: Simulated montage serial section image of the extruded Ti64-1.6B composite microstructure

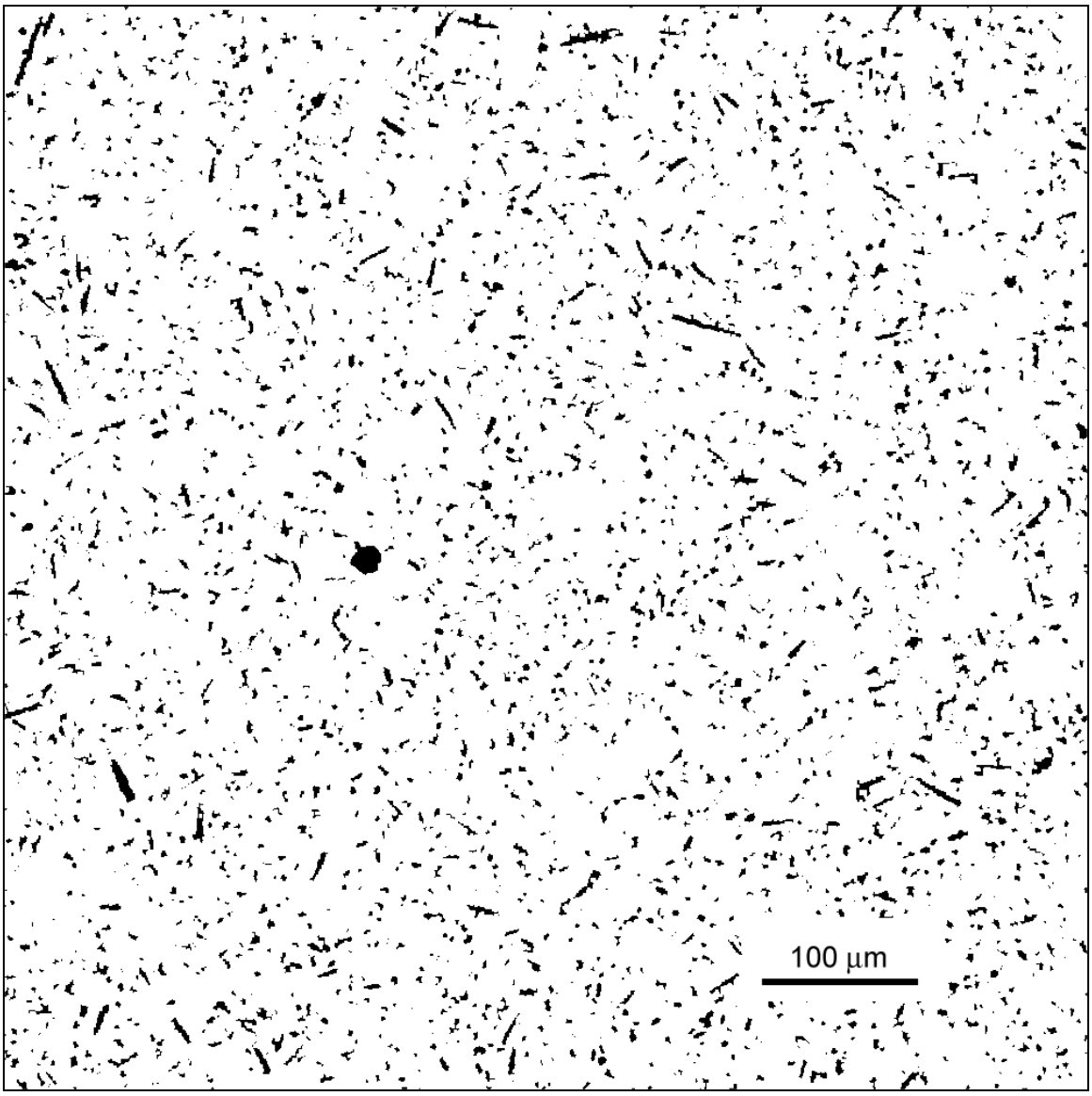


Figure 4.63: Simulated montage serial section image of the compacted Ti64-1.6B composite microstructure

The realistic nature of simulated microstructures has been validated via comparisons to the real microstructures using two-point correlation functions. These functions of the real and simulated microstructures in three perpendicular directions (Figures 4.64 to 4.69) show an excellent match. Short range matching of the correlation functions confirms the statistical similarity of whisker/particle sizes and shapes, whereas

the matching of the long-range part of the correlation functions confirms statistical similarity of the long-range heterogeneity of orientation distribution and spatial distributions of particles in the real and simulated microstructures.

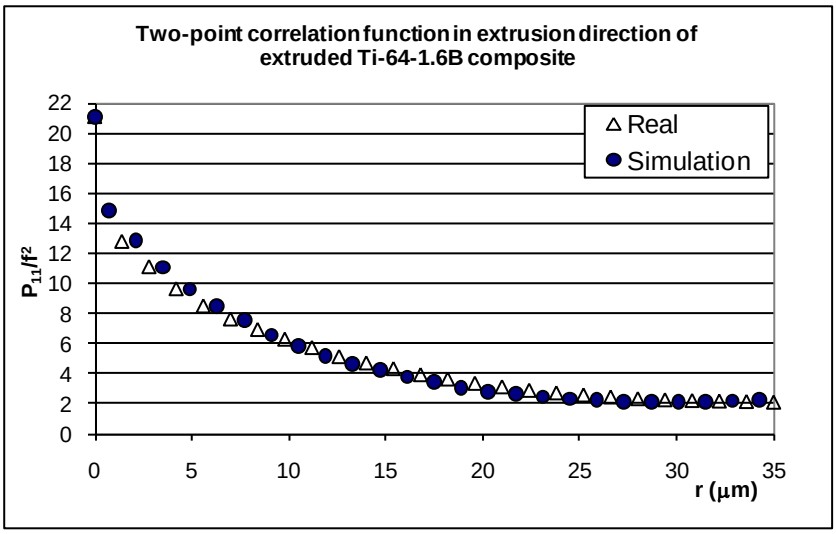


Figure 4.64: Comparison of two-point function in extrusion direction for extruded Ti64-1.6B composite microstructure

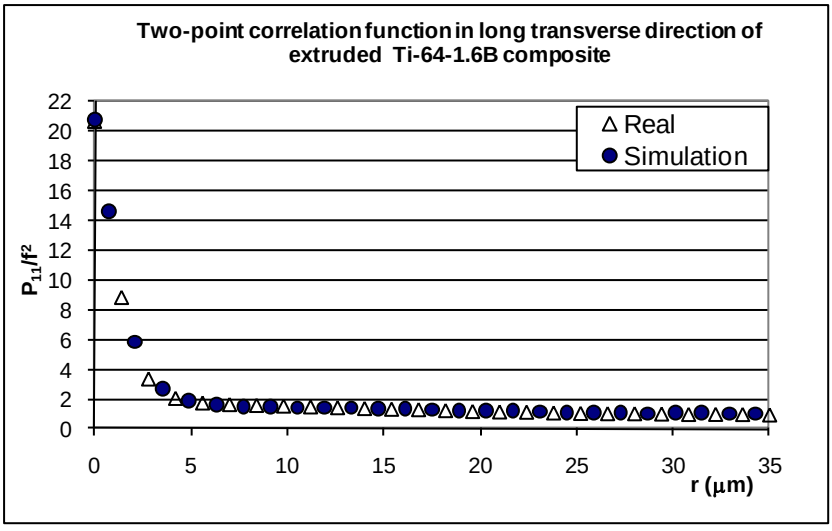


Figure 4.65: Comparison of two-point function in long transverse direction for extruded Ti64-1.6B composite real and simulated microstructures

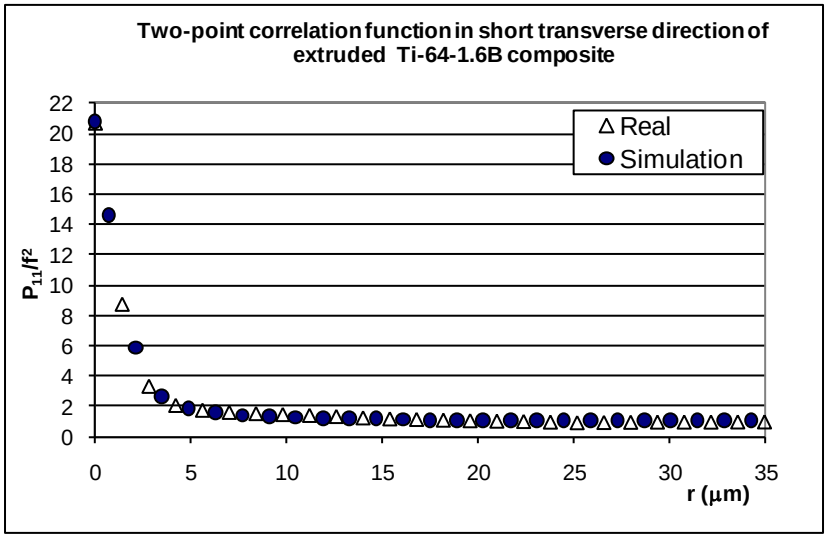


Figure 4.66: Comparison of two-point function in short transverse direction for extruded Ti64-1.6B composite real and simulated microstructures

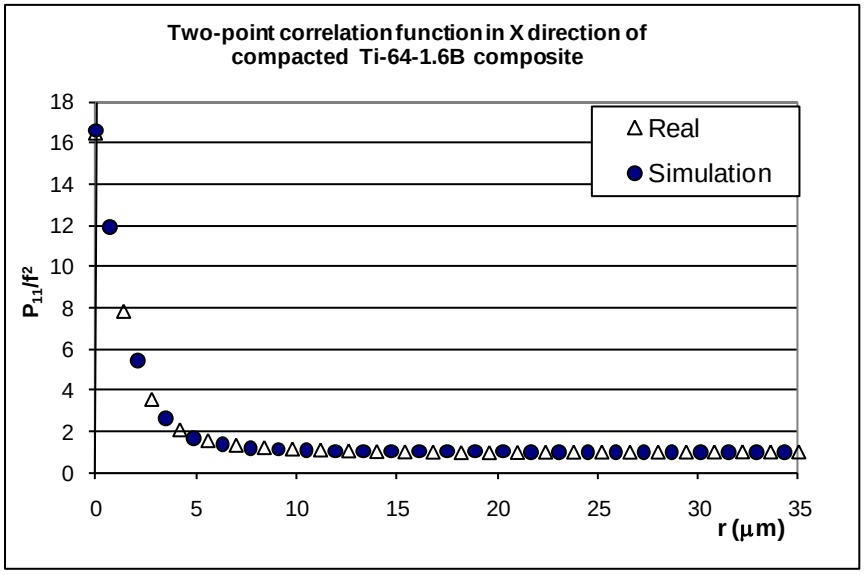


Figure 4.67: Comparison of two-point function in X direction for compacted Ti64-1.6B composite real and simulated microstructures

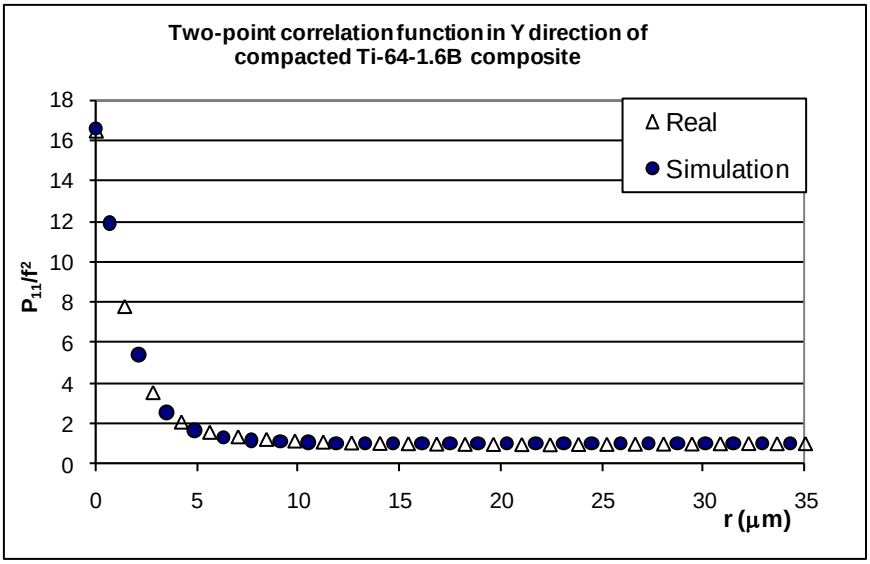


Figure 4.68: Comparison of two-point function in Y direction for compacted Ti64-1.6B composite real and simulated microstructures

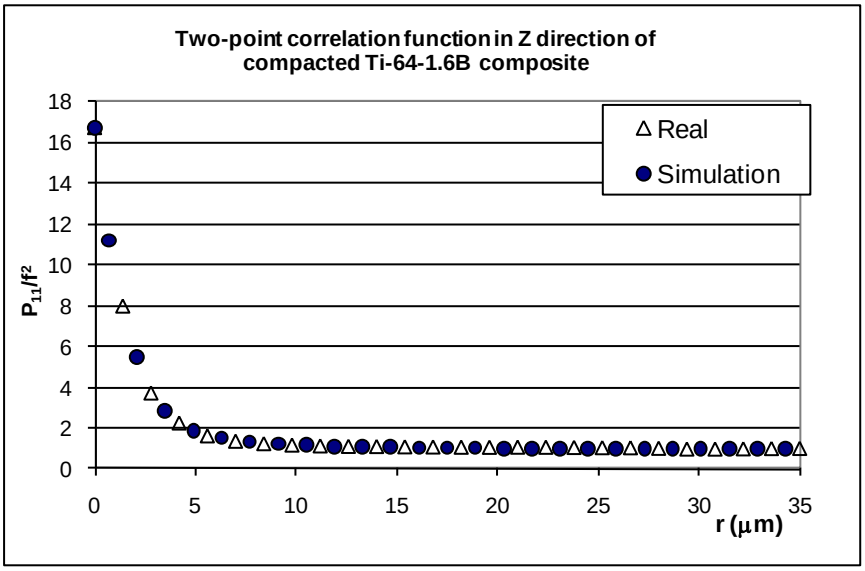


Figure 4.69: Comparison of two-point function in Z direction for compacted Ti64-1.6B composite real and simulated microstructures

4.3.6 Computer Simulated Virtual 3D Ti64-B Composite Microstructures

The methodology can be used to create 'virtual' microstructures of the composites that have not been fabricated by varying the numerical parameters in the model. Using the same set of TiB whiskers/particles, one can simulate a partially anisotropic microstructure by rotating the particles at a specified angle, which can represent changes in the microstructure due to deformation processing (e.g. extrusion) of the composite. Clearly, the amount of rotation given to each particle depends on the processing condition (e.g. the extrusion ratio and extrusion temperature). Figure 4.70 depicts a small segment of microstructural volume simulated by restricting x_1 and x_3 in Eq. (3) to the range [0, 0.1], which may be used to represent the partially anisotropic Ti64-1.6B composite extruded at a lower extrusion temperature and/or a lower extrusion ratio. Figure 4.71 shows a typical simulated 2D montage serial section of this simulated partially anisotropic Ti64-1.6B composite.

One can also simulate an atlas of virtual microstructures that have different TiB whiskers/particles volume fractions and different average particle sizes but the same spatial anisotropy. Figure 4.72 depicts a small 3D segment of one such simulation, where the whiskers have the same isotropic uniform random orientations as those in Figure 4.61 but the volume fraction of TiB whiskers is lower than that in Figure 4.61, and also the coarse primary TiB particles are excluded in the simulation. This simulated virtual microstructure may be used to represent a boron-modified hypoeutectic ($B \leq 1.55$ wt.%) Ti64-B alloy which only contain fine eutectic TiB whiskers. Figure 4.73 shows a typical simulated 2D montage serial section of this simulated hypoeutectic Ti64-B alloy. Additionally, one can simulate a virtual microstructure that is extruded from this virtual

isotropic microstructure by aligning the particles in extrusion direction as illustrated in Figures 4.74 and 4.75.

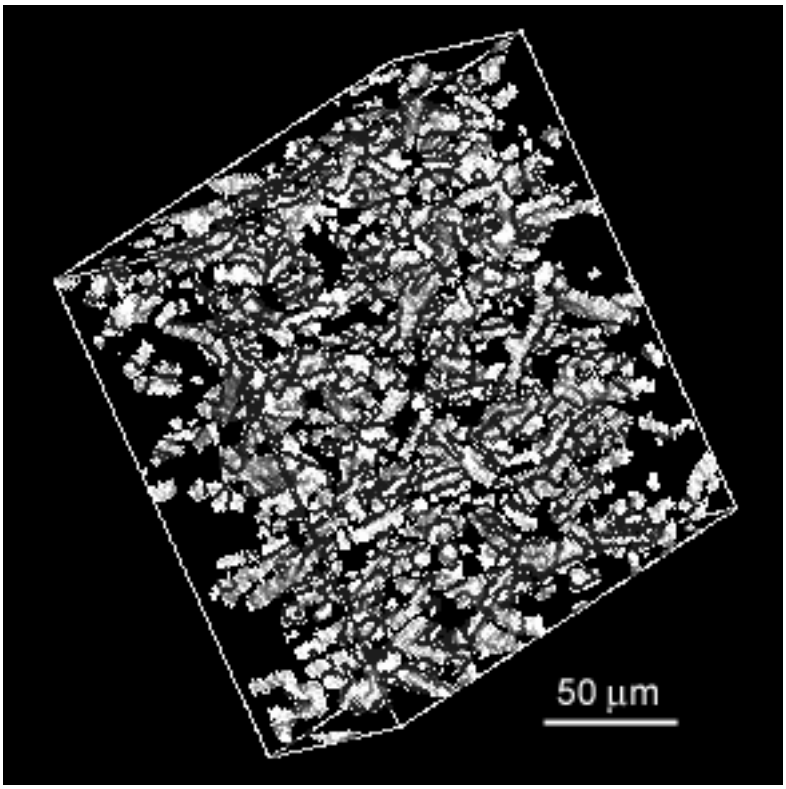


Figure 4.70: Small segment of virtual 3D microstructure of partially anisotropic Ti64-1.6B composite.

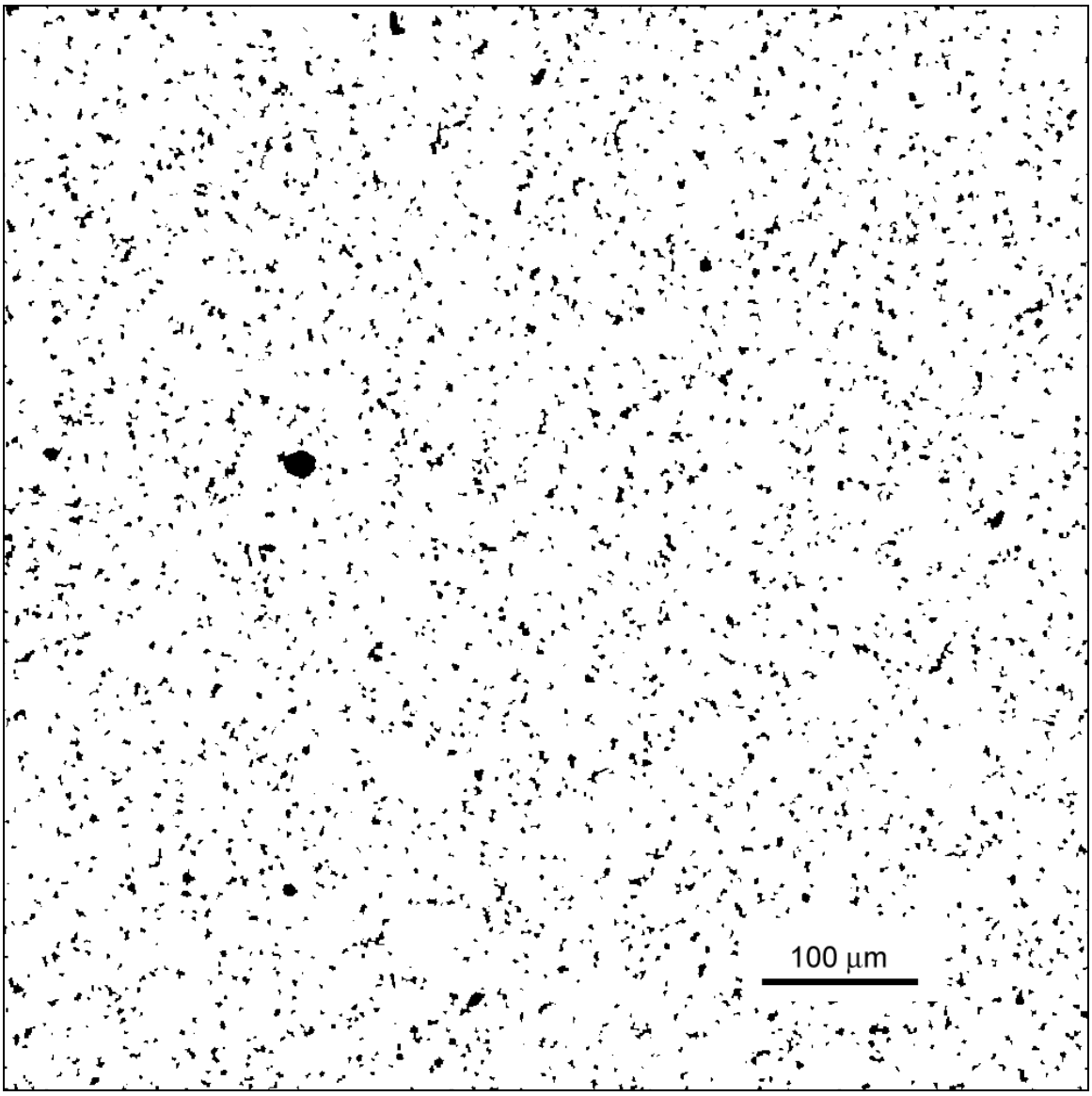


Figure 4.71: Montage of virtual 3D microstructure of partially anisotropic Ti64-1.6B composite.

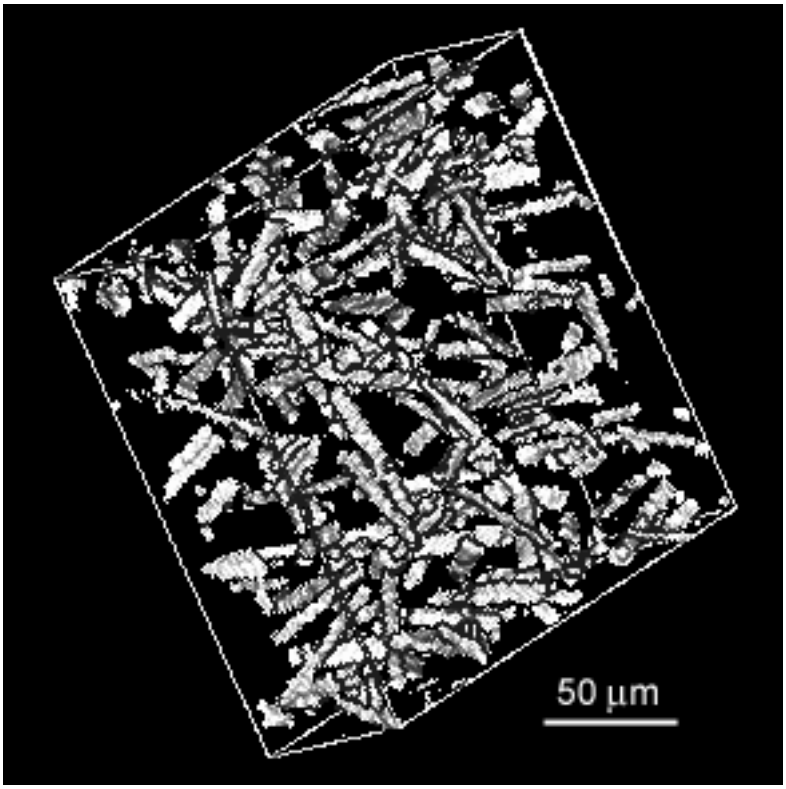


Figure 4.72: Small segment of virtual 3D microstructure of compacted hypoeutectic Ti64-B alloy

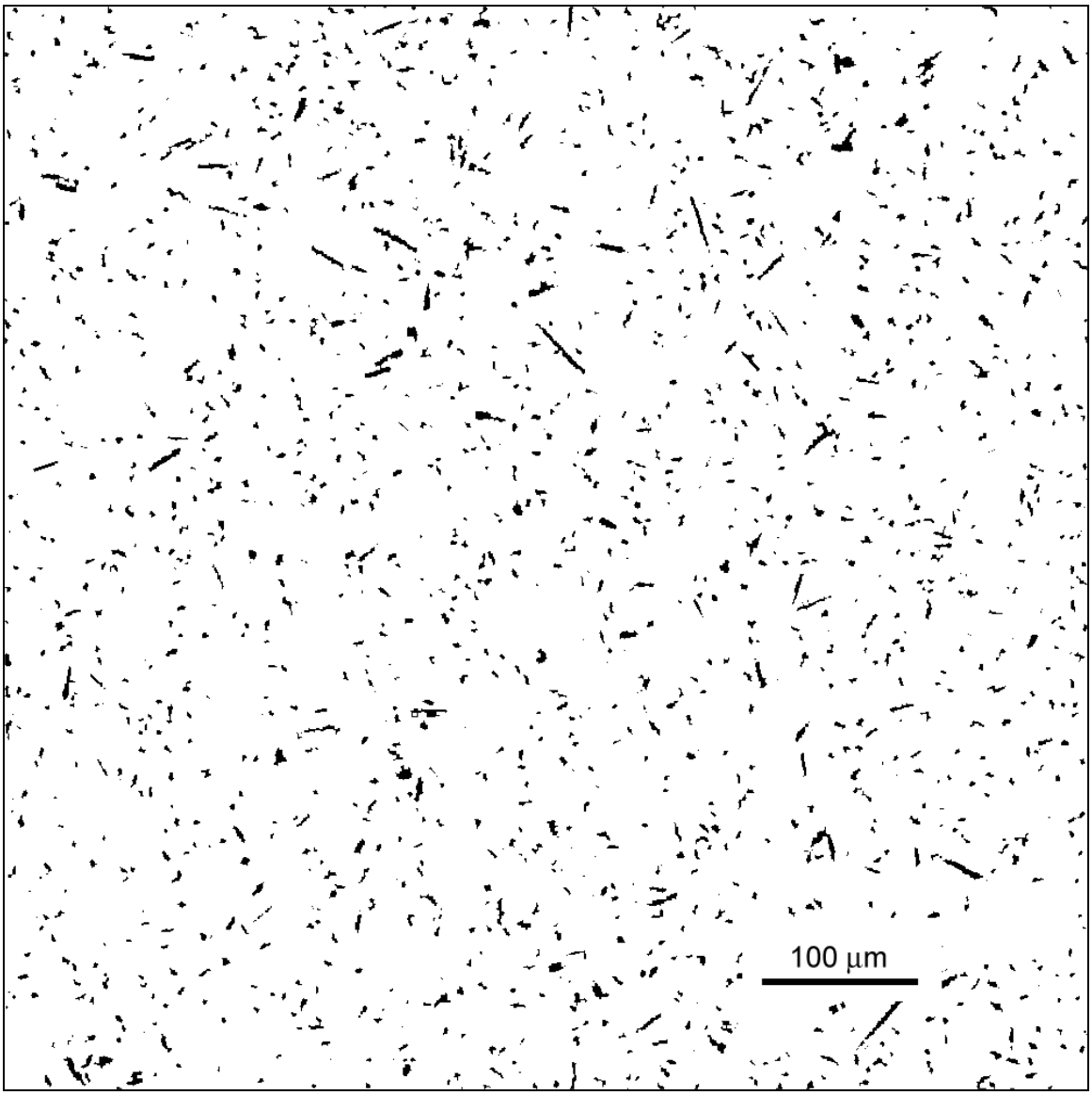


Figure 4.73: Montage of virtual 3D microstructure of compacted hypoeutectic Ti64-B alloy.

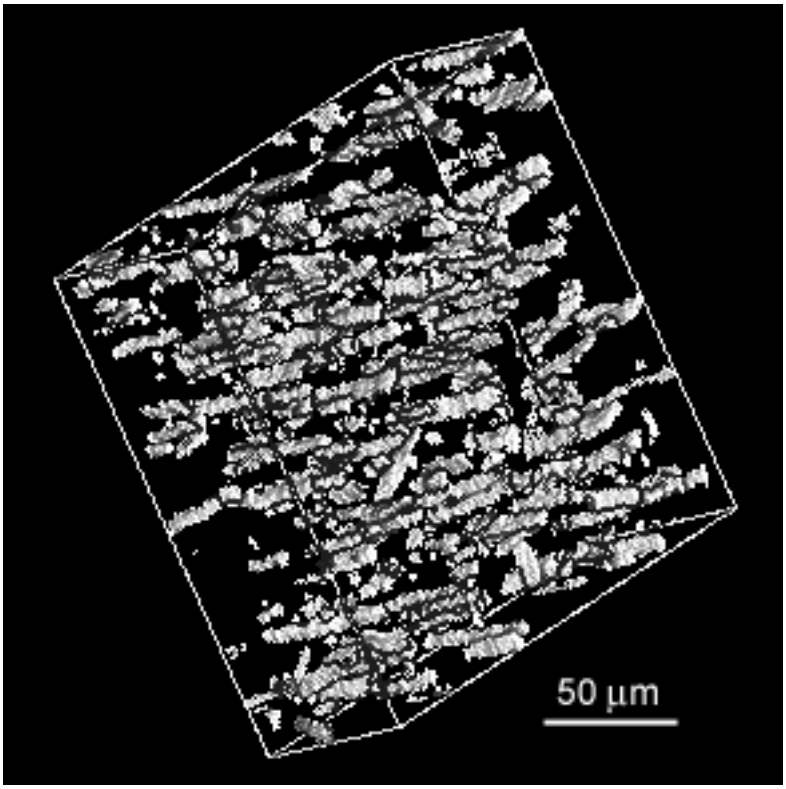


Figure 4.74: Small segment of virtual 3D microstructure of extruded hypoeutectic Ti64-B alloy.

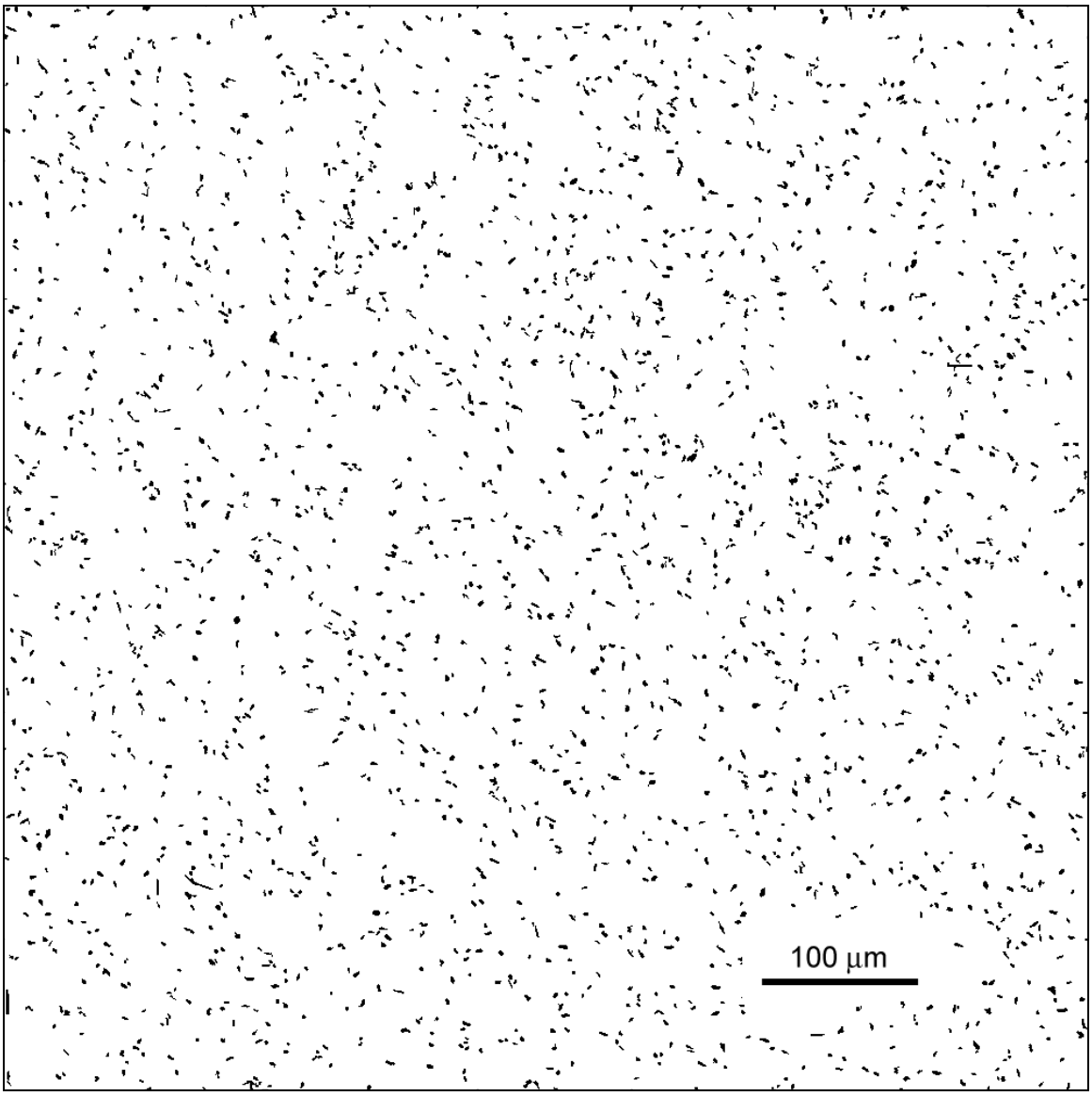


Figure 4.75: Montage of virtual 3D microstructure of extruded hypoeutectic Ti64-B alloy.

Finite elements (FE) based simulations can be carried out on these 3D virtual microstructures for realistic parametric studies on the effects of the microstructural parameters (e.g. weight percentage of boron) and processing routes on the mechanical response of Ti-B composites. The resulting data can provide useful information for optimization of the processing routes and materials design. These virtual microstructural

simulations can cut down on the number of experiments (and therefore, the time and resources) required for developing new composites and for optimizing the properties of the existing alloys.

4.4 Summary

A combination of digital image processing, stereology, stochastic-geometry-based microstructure representations, and well-known microstructure simulation algorithms has been utilized to develop a general and practical realistic 3D microstructure simulation technique for computer simulations of two-phase microstructures. The technique is presented through its application to the 3D microstructure of DRA composites containing SiC particles of complex shapes and boron modified Ti-6Al-4V composites containing fine TiB whiskers and coarse primary TiB particles.

The set of spatial (X, Y, Z) coordinates of the pixels/voxels comprising a distinct microstructural feature (a particle, whisker, etc.) in a 3D digital microstructure contains complete detailed information on the morphology and geometry of that feature. Once the information on the spatial coordinates of a microstructural feature is captured, an exact replica of that feature can be reproduced at any desired location in the simulation space. Therefore, such sets of (X, Y, Z) coordinate points of large number of features (~ few thousands) from representative digital images of a real microstructure can serve as a library of realistic feature images for simulations of the corresponding realistic microstructures. The feature images in the library can be then used to create a desired ensemble of particles/features having any other specified size distribution and volume fraction. Statistical descriptors such as the two-point correlation functions and lineal path probability functions for the simulated microstructure are computed and compared with the experimentally

measured functions for the real microstructure. The feature locations in the simulation are changed using Monte-Carlo based techniques till the statistical descriptors for the simulated microstructure are in agreement with that for the corresponding real microstructure. This leads to a realistic simulated microstructure having realistic complex feature morphologies similar to those in the corresponding real microstructures, and specified size distribution, spatial arrangement, and volume fraction of the microstructural features. The simulated microstructure is sufficiently large, so that short-range (on the order of particle/feature size) as well as long-range (hundred times the particle/feature size) microstructural heterogeneities and spatial patterns are represented at high resolution. The realistic nature of the simulated microstructure can be further confirmed by computing its mechanical responses using finite elements based simulations and comparing to the corresponding real microstructure. The simulation flowchart is given in Appendix A.1.

Once the simulation model is validated and the model parameters are correlated with processing parameters (such as reinforcement volume fraction, morphological orientations, size distribution, and extrusion ratio/temperature), it is possible to generate virtual microstructures of materials that have been processed differently than those from which the original data sets were obtained. The properties of such virtual materials can be computed through implementations of the simulated virtual microstructures in the FE-based simulations to obtain useful input for materials design and development.

CHAPTER 5

SURFACE AREA ESTIMATION BY DUAL-SCALE VIRTUAL CYCLOIDS

5.1 Introduction

To further develop a more realistic and sophisticated 3D materials microstructure modeling and simulation methodology, besides volume fraction, size distribution, orientation distribution, spatial clustering, and anisotropic, more geometric attributes (such as the surface area and mean curvature distribution, etc.) of the internal features are needed to incorporate in the microstructure models and simulations. Among these geometric attributes, the surface area distributions of 3D objects (such as grains, precipitates, voids, etc.) is one of the important geometric attributes of materials microstructure. The mechanical and physical properties of materials are often directly related to the surface areas of their internal objects. As most of the materials are opaque, the 3D objects are usually represented in a digitized form as a set of aligned 2D serial section images through the 3D volume. Clearly, these 2D digital section images do not contain all the information concerning a pre-digitized object's true 3D microstructural geometry. Therefore surface area of the 3D objects can only be estimated from these discrete data.

Numerous algorithms have been proposed to estimate the surface area of a digitized 3D object. Some approaches are based on local configuration counting [123-126], whereas others are based on global polyhedrization [127-128], and normal vector field integration [129-130]. Consider digital surfaces defined by sets of black and white

voxels in an adjacency grid. In the local configuration counting methods, first one determines the number of occurrences of certain patterns of black and white voxels in 2 × 2 × 2 neighborhood. An optimized weight is then assigned to each configuration of voxels and the total surface area is calculated by summation of the local area contributions. These local methods do not require a time consuming reconstruction of the actual object surface, which enable straightforward and efficient implementation. As the choice of the local configuration weights is not unique, a number of different choices have been investigated [123-126]. However, all proposed weights are optimized based on the regular cubic grid, which is not usually encountered in the raw volume images that are captured via serial sectioning technique, as the resolution within the section images is usually higher than the distance between the sections. Hence, before applying these local methods, the raw volume images need to be interpolated into the cubic grid. Another problem of these local methods is that they are not multigrid convergent [131]. The estimators do not converge to the true surface area as lattice resolution increases. On the other hand, a number of global polyhedrization based methods such as digital planar segment based polyhedrization [127] and relative convex hull [128] are multigrid convergent. These estimators converge to the true surface area as lattice resolution increases. They can be directly applied to non cubic grids. They are, however, computationally more intensive, as they require an explicit approximation of the boundary of the object. Coeurjolly et al. [129-130] have proposed a multigrid convergent method based on discrete normal vector field integration. However calculating gradients at all voxel positions and performing iterations in (large) spherical neighborhoods is not

computationally efficient. And because of the symmetry criterion used in the algorithm, this method cannot be directly applied to non-cubic grids.

As a different approach to the surface area estimation problem, stereologists estimate the surface area based on probability theory which involves statistical sampling of the 3D microstructure by test lines. The number of intersections between the test lines/probes and the surfaces of interest is counted. The population average value of the number of intersections between the test lines and the surfaces of interest per unit test line length, $<I_L>$, is related to the total surface area per unit volume, S_V, through the following stereological equation given by Smith and Guttman [132].

$$S_V = 2 <I_L> \qquad \text{Eq. 9}$$

This stereological method is efficient, multigrid convergent [133] and obviously can be directly applied to non-cubic grids. However, to obtain an unbiased estimation of surface area, isotropic uniform random (IUR) test lines and object interactions must be ensured which requires test lines in various orientations. This requirement renders the method problematic for digitized image volume as many orientations do not exist in discrete grids. Inspired by the need of an unbiased estimation of surface area of digitized 3D objects, this contribution presents a new stereological technique based on the relationship between two-point correlation functions and surface area. The theoretical development contributed by research colleague Shenjia Zhang is given in the following section.

Table 5.1: Comparison of Surface Area Estimators

Methods	Complexity	Multigrid Convergent	Non-Cubic Grids
Local Configuration Counting	O(n)	No	No
Global Polyhedrization	$O(n^2)$	Yes	Yes
Surface Normal	$O(n^2)$	Yes	No
Stereology	O(n)	Yes	Yes

5.2 Theoretical Development

5.2.1 The Relationship between Two-point Correlation Functions and Surface Area

Two point correlation function $P_{12}(r, \theta, \phi)$ and the intersection count in that direction $I_L(\theta, \phi)$ are related as follows [134]:

$$\frac{I_L(\theta,\phi)}{2} = \lim_{r\to 0}\frac{P_{12}(r,\theta,\phi)}{r} = P_{12}{}'(r,\theta,\phi)\Big|_{r=0} \qquad \text{Eq. 10}$$

Combining Eq. 9 and Eq. 10, the surface area can be defined as:

$$\frac{S_V}{4} = \lim_{r\to 0}\frac{< P_{12}(r) >}{r} = P_{12}{}'(r)\Big|_{r=0} \qquad \text{Eq. 11}$$

Where $<P_{12}(r)>$ is the average of $P_{12}(r, \theta, \phi)$ over all orientations (θ, ϕ) in the 3D space.

5.2.2 The Mathematical Modeling of Stereological Measurement of Surface Areas in Digital Images

The "intersection counting" technique is the most commonly used method to estimate the surface area of 3D microstructures with images of 2D sections. The technique can be applied to analogue micrographs as well as digital images. The digital version of the techniques is similar to the analogue version, except that periodically distributed discrete points in the path of the test probe (e.g. straight lines, cycloids) are used as actual test probes in lieu of continuous test probes.

Suppose a set of points that are a distance of r apart in the path of a straight line of length L is superimposed on a digital image of a 2D section of area A and the number of intersects I between the "test line" and the microstructural interface is counted by checking if the colors of pixels change from one test point to the next. Such procedure is

repeated N times. The intersection counting as described above is in essence an estimation of the 2-point correlation function at distance r.

$$P_{12}(r,\theta,\varphi) = \frac{Ir}{2NL} \qquad \text{Eq. 12}$$

The factor of ½ is introduced because the two-point correlation function is only evaluated in one direction:

$$P_{12}(r,\theta,\varphi) = \frac{Ir}{2NL} = P_{12}(r,-\theta,\varphi+\pi) \qquad \text{Eq. 13}$$

The surface area can be estimated by the following equation:

$$S_V^1 = 2\frac{I}{NL} \qquad \text{Eq. 14}$$

Compare it with Eq. 12, it is clear that

$$S_V^1 = 4\frac{P_{12}(r)}{r} \qquad \text{Eq. 15}$$

By comparing this relation with the Eq. 11, it can be shown that S_V^1 is the first order approximation of S_V by means of the Taylor expansion at $r = 0$.

The approximation is adequate in the case of "vertical" and "horizontal" test lines, since r is exactly equal to one pixel length for both cases. However, in order to obtain an unbiased estimation of surface areas in an anisotropic microstructure, test lines of various orientations are necessary. Due to the discrete nature of a digital image, it is not unusual that $r > 5$ for some of the orientations. For example, it can be shown that for an orientation of $\theta = \arctan(5)$ in a 2D plane, $r = \sqrt{1^2+5^2} \times PixelSize \approx 5.1 \times PixelSize$.

As r increases, the first-order approximation can be significantly biased, especially when the resolution along any direction is limited compared to the scale of the microstructural features under investigation. When a serial sectioning technique is used

to reconstruct the 3D microstructure, the inter-planar distance is usually much larger than the pixel size in the 2D sections, which sometimes makes the resolution perpendicular to the 2D planes (Z direction) comparable to the length scale of the features of interest. Inspired by the need of an unbiased estimation of surface in 3D microstructures, a modified technique is proposed. The general idea of the technique is to exploit the information contained in two-point correlation functions. The mathematical modeling is described as follows.

5.2.3 Mathematical Modeling of the Dual-scale Surface Area Estimation Technique

An unbiased estimation of surface area can be achieved by extrapolating the two-point correlation function to the origin as $r \rightarrow 0$. Let us approximate the $P_{12}(r)$ in the vicinity of $r = 0$ with a third-order polynomial.

$$P_{12}(r) \approx a_0 + a_1 r + a_2 r^2 + a_3 r^3 \qquad \text{Eq. 16}$$

Frisch and Stillinger [135] have shown that $a_2 = 0$ for all two-point correlation functions when the surfaces contains no singular points at which the radii of convergence of the canonical expansion shrink to zero. Since $P_{12}(0) = 0$, a_0 is also zero. Let us define $\widehat{P}_{12}(r)$ as the approximated two-point correlation function near $r = 0$. We have

$$\widehat{P}_{12}(r) = a_1 r + a_3 r^3 \qquad \text{Eq. 17}$$

The estimator contains two coefficients, which requires at least two equations to determine. Two equations can be constructed by performing the "intersection counting" twice at different point spacing: r and $2r$. The directly measured surface areas are:

$$S_V^1 = 4\frac{P_{12}(r)}{r} \qquad \text{Eq. 18}$$

$$S_V^2 = 4\frac{P_{12}(2r)}{2r} \qquad \text{Eq. 19}$$

Combine Eq. 17, Eq. 18 and Eq. 19, it follows that

$$\begin{cases} \frac{1}{4} r S_V^1 = a_1 r + a_3 r^3 \\ \frac{1}{8} r S_V^2 = 2a_1 r + 8a_3 r^3 \end{cases} \qquad \text{Eq. 20}$$

Solve the equation,

$$a_1 = \frac{1}{4}\left(\frac{4}{3} S_V^1 - \frac{1}{3} S_V^2\right) \qquad \text{Eq. 21}$$

Applying Eq. 11,

$$\widehat{S}_V = 4\widehat{P}_{12}{}'(r)\Big|_{r=0} = 4a_1 \qquad \text{Eq. 22}$$

Therefore,

$$\widehat{S}_V = \frac{4}{3} S_V^1 - \frac{1}{3} S_V^2 \qquad \text{Eq. 23}$$

The equation above gives an estimator $\widehat{S}_V$ for the actual surface area of a 3D microstructure.

5.2.4 The Application of Dual-scale Estimation to Virtual Cycloids

Virtual cycloids can be used to obtain unbiased estimation of surface area of anisotropic microstructures with higher efficiency than line segments since the special geometric shape of the curves is designed to measure the projected surface areas in all orientation with correct weighing in a spherical coordinate system [136].

To use cycloids as probes in digitally reconstructed microstructures, discrete versions of cycloids are used. As illustrated in Figure 5.1, the digitization process substitutes the continuous curve by a set of test points which coincides with pixel positions on the cycloid. The discrete cycloids are therefore equivalent to a set of test line segments *dr* with various orientations having lengths directly proportional to $\sin\theta$.

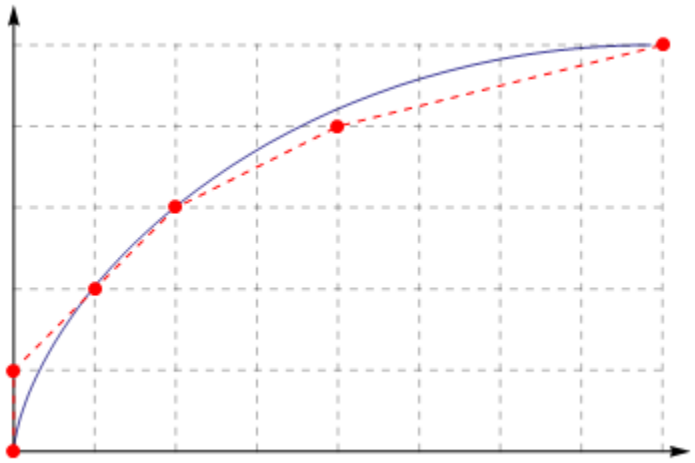


Figure 5.1: A cycloid (blue solid line) and a discrete cycloid (red dashed line)

To apply the dual-scale method to cycloids, an ideal model of cycloid-estimated surface area is given by

$$S_V = \frac{4}{L}\int_0^L P_{12}^{'}(\theta,\varphi,r)\Big|_{r=0} dr \qquad \text{Eq. 24}$$

Where L is the length of the cycloid, *dr* is a line segment on the cycloid and it is equal to $L\sin(\theta)d\theta$.

Due to the digitization of the cycloid, not all possible orientations are achievable and therefore a piece-wise segmentation approximation is given by

$$\tilde{S}_V = \frac{4}{L}\sum_i (P_{12}^{'}(\theta_i, \phi_i, r)\big|_{r=0})r_i \qquad \text{Eq. 25}$$

Apply Eq. 22 to get the following result,

$$\hat{S}_V = \frac{4}{L}\sum_i a_1^{(i)} r_i \qquad \text{Eq. 26}$$

Where $a_1^{(i)}$ is the coefficient of the two-point function approximation corresponded to i^{th} segment of the discrete cycloid.

The measurement result with dual-scale cycloid probes can be interpreted as

$$S_V^1 = \frac{4}{L}\sum_i P_{12}(\theta_i, \phi_i, r_i) \qquad \text{Eq. 27}$$

and ,
$$S_V^2 = \frac{4}{2L}\sum_i P_{12}(\theta_{i,}, \phi_i, 2r_i) \qquad \text{Eq. 28}$$

Apply Eq. 17, it follows

$$S_V^1 = \frac{4}{L}\sum_i (a_1^{(i)} r_i + a_3^{(i)} r_i^3) \qquad \text{Eq. 29}$$

and ,
$$S_V^2 = \frac{4}{2L}\sum_i (2a_1^{(i)} r_i + 8a_3^{(i)} r_i^3) \qquad \text{Eq. 30}$$

Solve the equations and it can be shown that

$$\sum_i a_1^{(i)} r_i = \frac{1}{4}\left(\frac{4}{3}S_V^1 - \frac{1}{3}S_V^2\right) \qquad \text{Eq. 31}$$

Then it follows that

$$\hat{S}_V = \frac{4}{3}S_V^1 - \frac{1}{3}S_V^2 \qquad \text{Eq. 32}$$

The Eq. 32 shows that the dual-scale method can be directly applied to cycloid probes. The only difference in the implementation is the usage of discrete cycloids instead of line segments.

5.3 Implementations of Dual-scale Virtual Cycloids

The original concept of placing virtual cycloids on the vertical sections to estimate surface area of an arbitrary orientated 3D object is given by Gokhale et al. [136]. Based on the vertical section approach of Baddeley et al. [137], the original virtual cycloids approach specifies the sectioning direction (i.e. the direction perpendicular to the 2D serial section images, namely Z direction) as vertical axis, and applies computer-generated virtual cycloids (as illustrated in Figure 5.2) with their minor axes parallel to the vertical axis. The number of surface-cycloid intersections counted on the serial section planes though the Z direction is proportional to the surface area of the 3D object of interest, with no further assumptions about size, shape or orientation of the 3D object.

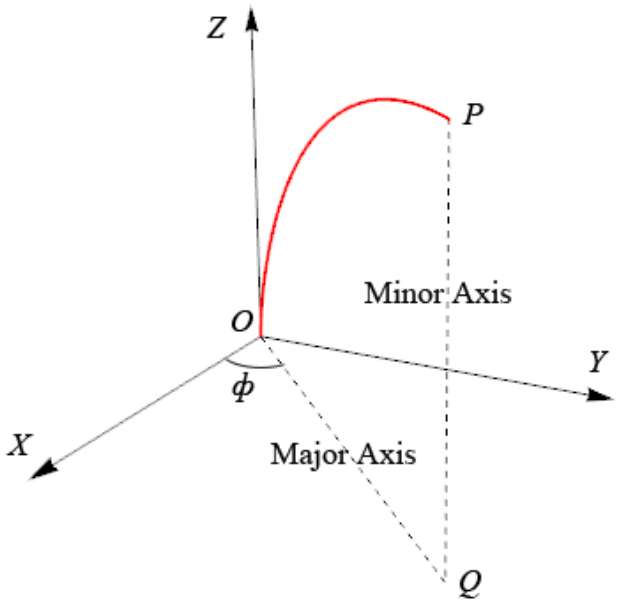


Figure 5.2: A cycloid with its minor axis (PQ) aligned to the Z axis. The orientation of the cycloid is defined as the angle ϕ between its major axis (OQ) and X axis.

The parametric equation for a cycloid is

$$\begin{cases} x = r(\theta - \sin\theta)\cos\phi + x_0 \\ y = r(\theta - \sin\theta)\sin\phi + y_0 \\ z = r(1 - \cos\theta) + z_0 \end{cases} \quad \text{Eq. 33}$$

Where parameter θ is in the range of $0 \le t \le \pi$, (x_0, y_0, z_0) is the starting point of the cycloid, 4r is the length of the cycloid, and ϕ is the orientation of cycloid.

Dual-scale virtual cycloids approach estimates the surface area with the same concept as described above. S_V^1 is estimated by placing cycloids inside the digital volume constructed by 2D serial section images of the 3D object of interest and counting the intersection between cycloids and the surface of the 3D object. For estimation of S_V^2, the digital volume is first resampled at the half of the original resolution, and then the same set of the cycloids are placed in the downsampled digital volume to count the surface-cycloids intersections. The detailed procedure is described as follows.

Consider an example of utilizing virtual cycloid to estimate S_V^1 of a sphere which is sliced by six 2D sections, as illustrated in Figure 5.3. The virtual cycloid's minor axis is parallel to the Z axis which is the direction perpendicular to the 2D sections. When the virtual cycloid intersects a plane, it appears as a point (the red dot as shown in Figure 5.4). This point moves through the 2D sections according to the following equation which is derived by eliminating t in Eq. 33.

$$\begin{cases} x = (r\cos^{-1}(1 - \dfrac{(z - z_0)}{r}) - \sqrt{(z - z_0)(2r - z + z_0)})\cos\phi + x_0 \\ y = (r\cos^{-1}(1 - \dfrac{(z - z_0)}{r}) - \sqrt{(z - z_0)(2r - z + z_0)})\sin\phi + y_0 \end{cases} \quad \text{Eq. 34}$$

Where (x, y, z) is the new position of the cycloid intersection.

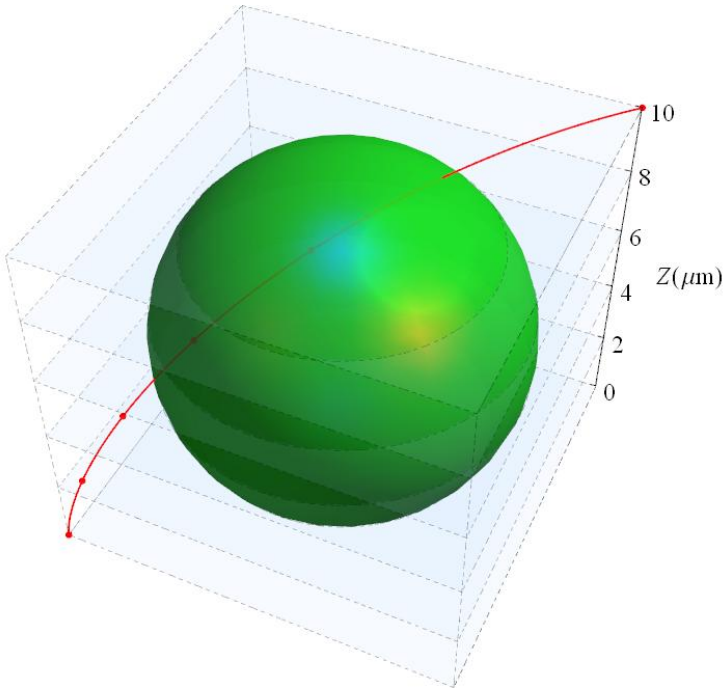


Figure 5.3: A virtual cycloid (red line) placed in a 3D volume containing a sphere (green ball), which are sliced into six 2D serial sections displayed in Figure 5.4.

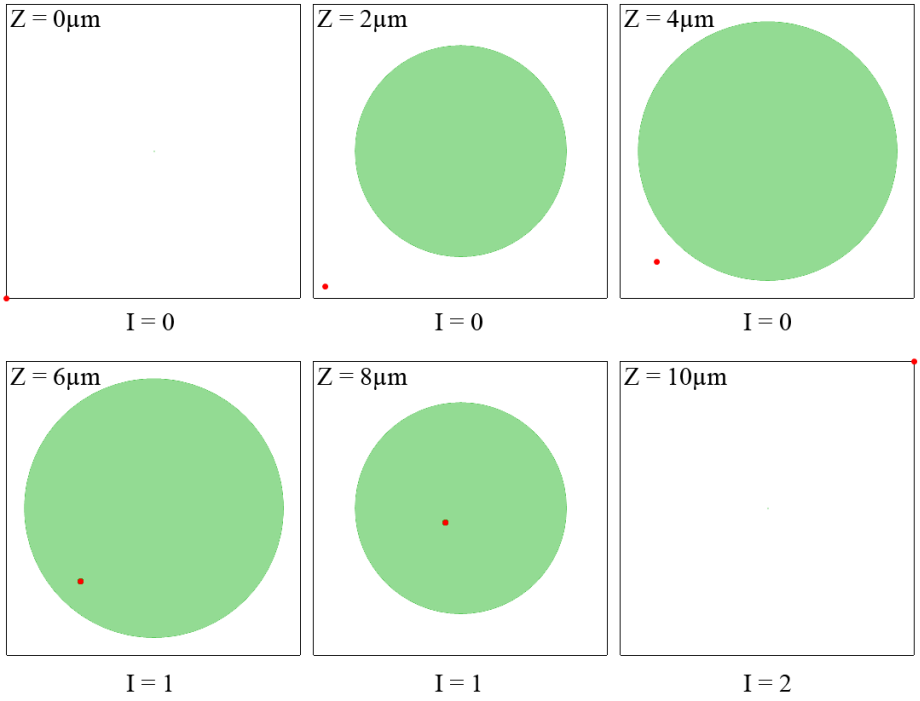


Figure 5.4: Six serial section images depict the relationship between cycloid cross-section (red dot) and sphere cross-sections (green disk). The cycloid-surface intersection count number $I = 2$.

If the cycloid point resides within a feature of interest (the green disk in the current example) in a given 2D section image, and the cycloid point appears outside the feature in the next plane, then obviously the cycloid has crossed the interface of interest once between these two adjacent planes (i.e. one intersection count). Therefore if one tracks the number of times the cycloid point goes in and out of the feature of interest, then that number is equal to the intersection count with the corresponding virtual cycloid. This can be done automatically by tracking the change of 2D section image color along the path of the cycloid. As shown in Figure 5.4, the color of the cycloid intersections changes from white to green to white. This results an intersection count of 2.

By placing virtual cycloids in the 2D section images at systematic-random locations and at systematic-random rotations around the vertical axis for efficient sampling, and using the automatic intersection counting technique, a sufficient number of intersections can be counted automatically. S_V^1 can then be calculated using Eq. 9.

Once S_V^1 is calculated, the digital volume is resampled at the half of the original resolution, and then the same set of the cycloids used to estimate S_V^1 are placed in the downsampled digital volume to count the number of surface-cycloids intersections. S_V^2 can then be calculated using Eq. 9. Eq. 32 is then used to estimate surface area per unit volume ($\hat{S}_V$). Finally, the surface area ($\hat{S}$) of the 3D object is estimated as the product of $\hat{S}_V$ and total volume (V) of the sampling space:

$$\hat{S} = \hat{S}_V \times V \qquad \text{Eq. 35}$$

The flow chart and computer code for dual-scale virtual cycloids are given in Appendix A.2 and B.9. To evaluate the performance of the dual-scale virtual cycloids,

the estimator has been applied to a set of convex and non-convex 3D objects of known surface area with various isotropic and anisotropic morphological orientations. The results are reported in the following section.

5.4 Results and Discussion

Five different types of 3D objects have been used to evaluate the performance of the dual-scale virtual cycloids, as illustrated in Figure 5.5. They are (a) balls with increasing diameters; (b) disk shaped ellipsoids with increasing sizes where the ratio between three radii is a:b:c = 10:10:1; (c) needle shaped ellipsoids with increasing sizes where the ratio between three radii is a:b:c = 10:1:1; (d) ring tori with increasing sizes where the distance from the center of the tube to the center of the torus (R) is 2 times the radius of the tube (r) and (e) horn tori with increasing sizes where the distance from the center of the tube to the center of the torus (R) is equal to the radius of the tube (r). The five sets of 3D objects were generated in the continuous space and digitized in different sizes and positions. Ellipsoid and torus shaped objects are digitized with three different orientations, which are (i) major axis aligned to the X axis, (ii) major axis aligned to the Z axis, (iii) randomly orientated.

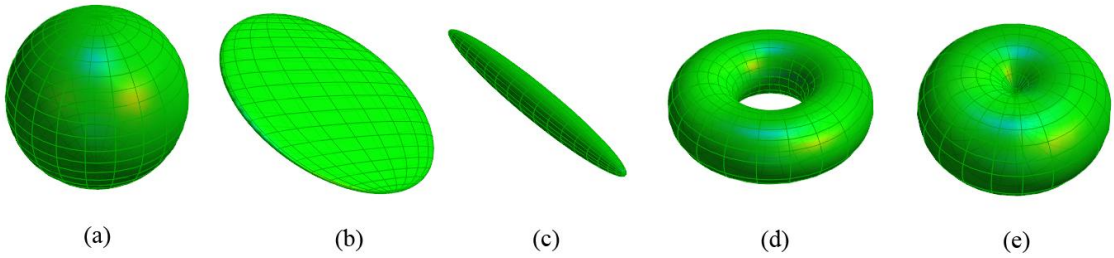


Figure 5.5: Five different shapes of objects, (a) a ball (b) a disk shaped ellipsoid (a:b:c = 10:10:1), (c) a needle shaped ellipsoid (a:b:c=10:1:1), (d) a ring torus (R = 2r), (e) a horn torus (R = r).

In order to mimic the raw serial section images that are captured by commonly used serial sectioning technique, all digitized objects are resampled by slicing the object into 8 2D section images at equal distance from each other along the Z axis while keep the resolution in XY plane at 1 pixel. Note that by sampling in this way, for large size objects the resolutions within the section images are much higher than the distance between sections, which is similar to that encountered in physical serial sections. Also note that different sets of these 8 2D section images can be sliced from the same object by changing the starting position of the first slice.

Once the section images are generated, 50,000 dual-scale virtual cycloids are then used to estimate the surface area. Figure 5.6 shows the maximum error of the surface area estimated for digitized balls with increasing diameters by dual-scale virtual cycloids which are compared to the maximum error estimated by original virtual cycloids. The results clear show improvements in accuracies. The dual-scale virtual cycloids estimated surface area converges to the true value much faster.

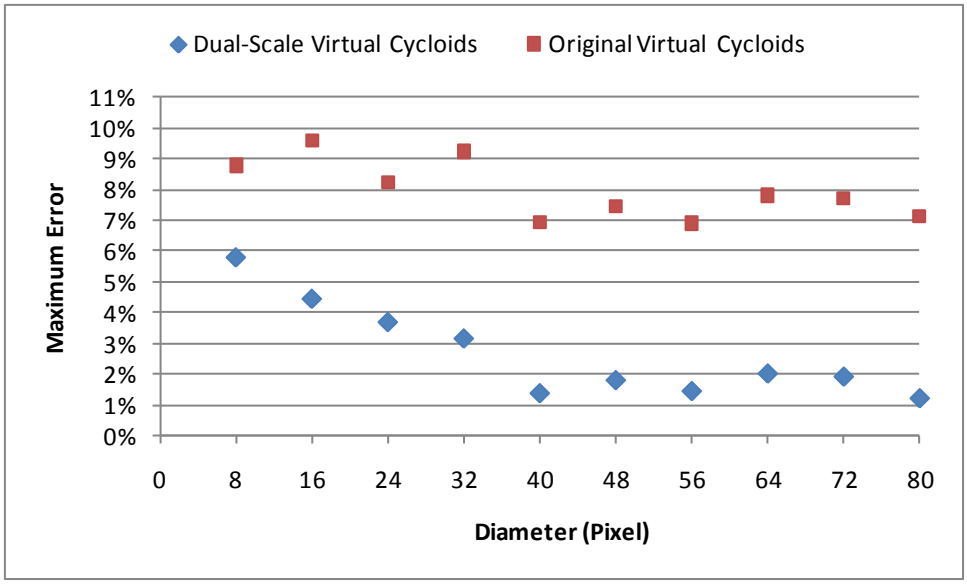


Figure 5.6: Comparison of maximum error of the surface area estimated for balls with increasing diameters by dual-scale virtual cycloids and original virtual cycloids

Figures 5.7 and 5.8 show the maximum error of the surface area estimated for ellipsoids and tori with increasing sizes and different orientations. The results show a similar trend of fast convergence to the true value. With just 8 voxels in object's size, the maximum error for the surface area is generally below 6%. Increasing the size of object (or in other words, increasing the resolution of 2D section image) to 24, while keep the resolution in Z direction (i.e. 8 2D section images), reduces the maximum error to 3%. This suggests that higher accuracy can be achieved efficiently by taking higher resolution 2D section images without the need for finer serial sectioning.

With the performance of the surface area estimator verified on convex and non-convex, isotropic and anisotropic orientated digitized simulated 3D objects, the dual-scale virtual cycloid have been utilized to characterize some complex microstructures in the cast Al-Si base alloys [138], as shown in Figure 5.9.

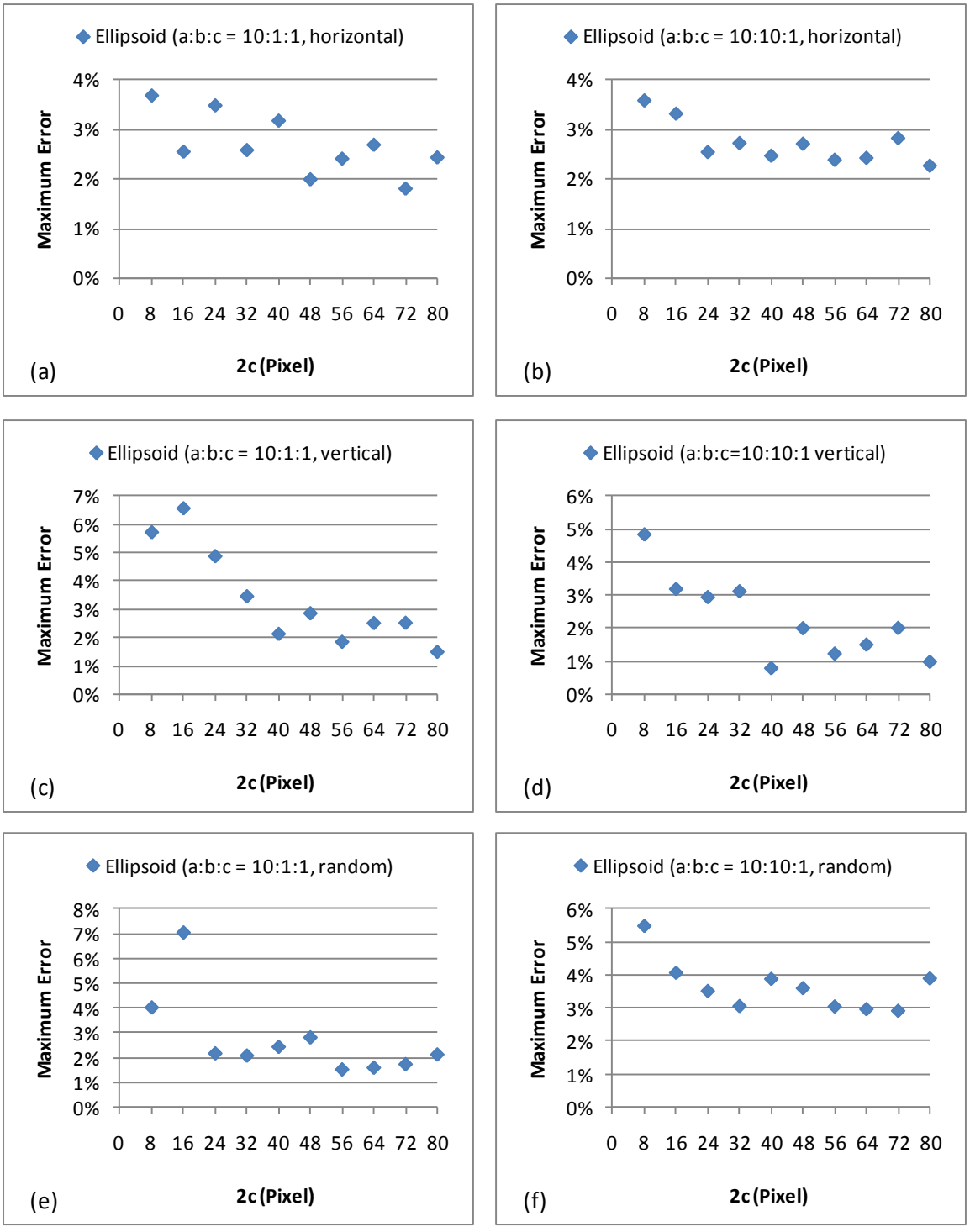


Figure 5.7: Maximum error of the surface area estimated for different types and orientations of ellipsoids with increasing sizes by dual-scale virtual cycloids. Needle shaped ellipsoid with its major axis (a) aligned to the X axis, (c) aligned to the Z axis, (e) randomly orientated; Disk shaped ellipsoid with its major axis (b) aligned to the X axis, (d) aligned to the Z axis, (f) randomly orientated.

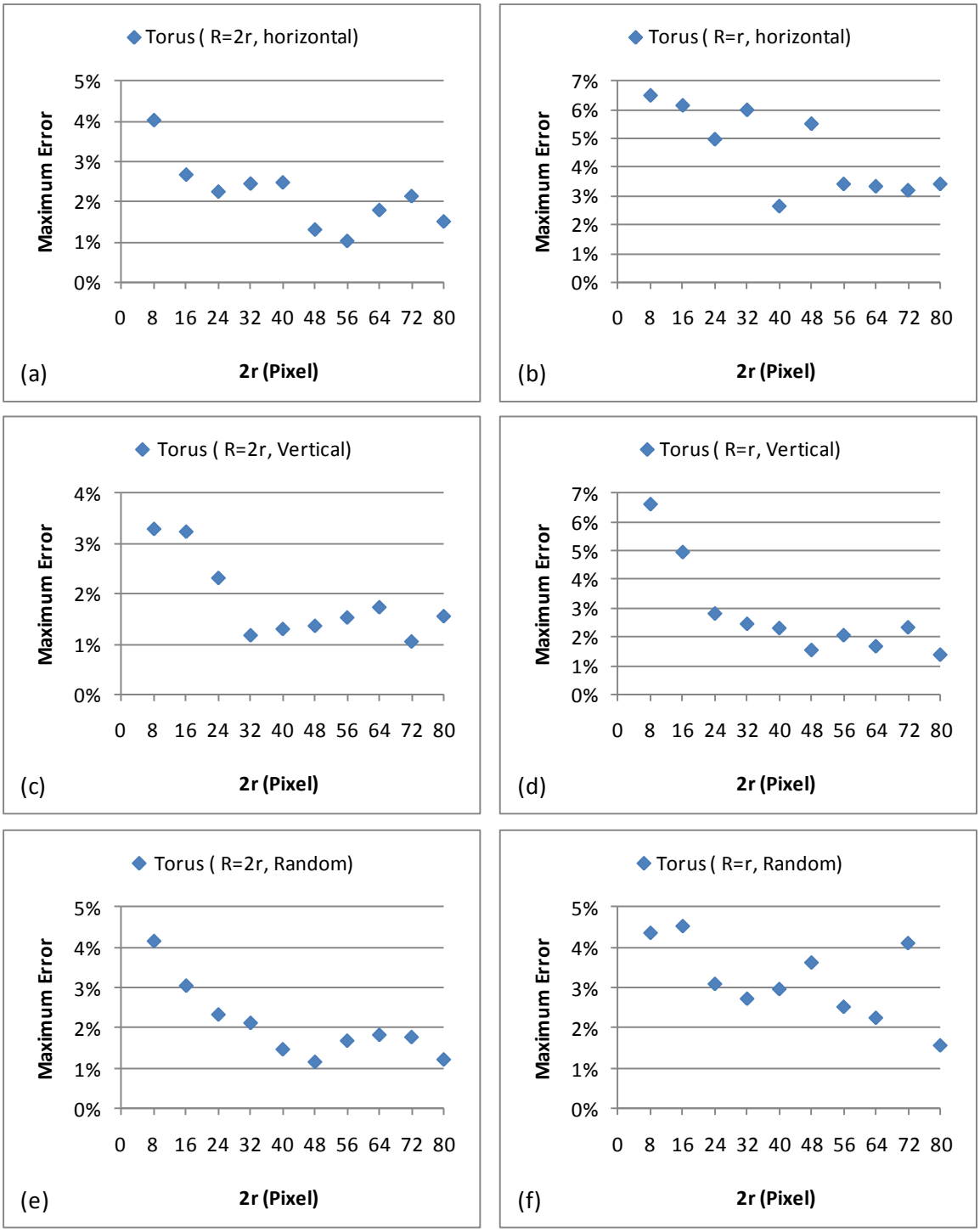


Figure 5.8: Maximum error of the surface area estimated for tori with increasing sizes by dual-scale virtual cycloids. Ring torus with its major axis (a) aligned to the X axis, (c) aligned to the Z axis, (e) randomly orientated; Horn torus with its major axis (b) aligned to the X axis, (d) aligned to the Z axis, (f) randomly orientated.

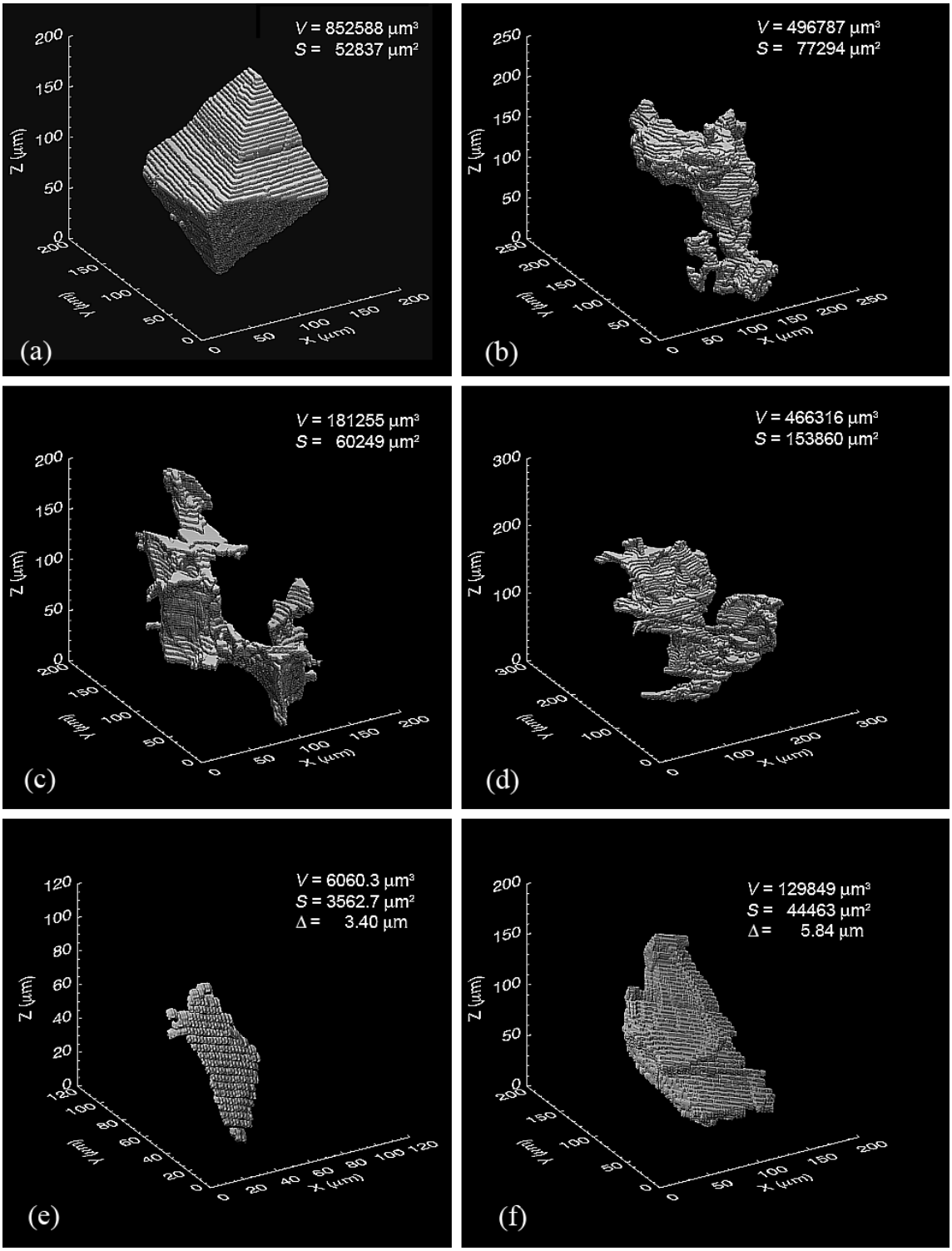


Figure 5.9: Examples of surface area of multiple phases in a cast Al-Si based alloy estimated by dual-scale cycloids: (a) Primary Si particle; (b) Gas pore; (c and d) Script intermetallics of convoluted complex 3D morphologies; (e and f) eutectic Si platelets. The resolution of the 2D serial section images is 0.3 µm, while the average distance between each section image is 3.2 µm.

5.5 Summary

In this chapter, a new unbiased and assumption free method for estimating surface area of digitized 3D objects constructed from 2D serial section images using dual-scale virtual cycloids has been presented. The approach is based on the relationship between two-point correlation functions and surface area. It specifies the sectioning direction (i.e. the direction perpendicular to the 2D serial section images) as vertical axis, and applies computer-generated virtual cycloids to both original 2D section images and downsampled 2D section images with their minor axes parallel to the vertical axis at systematic-random locations and at systematic-random rotations around the vertical axis. The number of surface-cycloid intersections is counted on the original and downsampled 2D serial section images though the Z direction. The surface area is then given by Eq. 9 and Eq. 32. The new estimator does not require a time consuming reconstruction of the 3D object surface, which enables straightforward and efficient implementation. The performance and accuracy of the new surface area estimator are verified on convex and non-convex, isotropic and anisotropic digitized objects.

CHAPTER 6

SUMMARY AND RECOMMENDATIONS FOR FUTURE RESEARCH

6.1 Summary

Modeling and simulations of microstructures at length scales of interest is an important aspect of computational materials science. However, almost all the two- and 3D geometric simulations of microstructures reported in literature utilize idealized simple particle shapes or incorporate unrealistic arbitrary digitized particle shapes, and/or assume uniform-random spatial arrangement of the features, and/or assume isotropic (or completely anisotropic) morphological orientations of the microstructural features. Such microstructure simulations are not likely to be useful for quantitative predictions of the mechanical and physical properties of complex real material microstructures. Therefore there is a need to develop techniques for simulations of realistic complex 3D microstructures that (1) incorporate realistic complex particle/feature shapes, (2) allow controlled non-uniformities/clustering in spatial distributions of features, (3) permit partial anisotropic morphological orientations of microstructural features, (4) closely match experimentally measured attributes of the corresponding real microstructures, and (5) efficiently generate sufficiently large segments of microstructure that contain short-range, intermediate-range and long-range microstructural heterogeneities and spatial patterns. The present research utilizes a combination of digital image processing techniques and computer simulations to develop an efficient and general methodology for representation and simulations of such realistic two- and three-dimensional

microstructures. The simulations incorporate realistic 2D as well as 3D complex morphologies/shapes, spatial patterns, anisotropy, volume fractions, and size distributions of the microstructural features statistically similar to those in the corresponding real microstructures. The methodology permits simulations of sufficiently large 2D and 3D microstructural windows that incorporate short-range (on the order of particle/feature size) as well as long-range (hundred times the particle/feature size) microstructural heterogeneities and spatial patterns at high resolution. The utility of the technique has been successfully demonstrated through its application to the 2D microstructures of constituent particles in wrought Al-alloys, the 3D microstructure of a discontinuously reinforced Al-alloy (DRA) composite containing SiC particles that have complex 3D shapes/morphologies and spatial clustering, and 3D microstructure of the boron modified Ti-6Al-4V composites containing fine TiB whiskers and coarse primary TiB particles. The simulation parameters are correlated with materials processing parameters (such as reinforcement volume fraction, morphological orientations, size distribution, and extrusion ratio/temperature), which makes possible the simulations of rational virtual 3D microstructures for the parametric studies on microstructure-properties relationships. The simulated and virtual microstructures have been implemented in the 3D finite-elements (FE)-based framework for simulations of micro-mechanical response and stress-strain curves. Finally a new unbiased and assumption free dual-scale virtual cycloids probe for estimating surface area of 3D objects constructed by 2D serial section images is also presented.

6.2 Recommendations for Future Research

In the present research, two-point correlation functions and lineal path probability distribution functions have been used for microstructure representation. It follows that the corresponding simulated microstructure can mimic only those aspects of real microstructural geometry that are implicitly (or explicitly) represented by the two-point correlation functions and lineal path functions. Nonetheless, the simulation methodology is quite general, and therefore it can also be used to simulate microstructures having specified higher order correlation functions (for example, three-point and four-point correlations), if needed. However, at present, the experimental techniques for efficient and robust estimation of the higher order correlation functions of 3D microstructures are not well developed, and therefore, more research of on higher order microstructural correlation functions is needed.

The parameters in the present simulation models have been varied to create 'virtual' microstructures that represent changes in the processing parameters of these materials. In the present research, the simulated virtual microstructures are limited to have a range of different constituent particle volume fractions and spatial clustering. Nonetheless, the simulation methodology is quite flexible, and therefore it is possible to simulate virtual microstructures having any specified particle size distribution, surface area distribution, and/or orientation distribution, etc. More research of such parametric studies is needed.

In its present form, the methodology for computer simulations of realistic 2D and 3D microstructures is primarily applicable to two-phase microstructures only. Nonetheless, many material microstructures are multi-phase (more than two phases). Therefore, there is a need to further develop the current methodology for simulations of realistic multiphase 3D microstructures where the feature morphologies/shapes, volume

fractions, size distributions, spatial patterns and anisotropy are statistically similar to those in the corresponding real microstructures. Some preliminary work has already been done in this direction. Figures 6.1 and 6.2 show one such multiphase simulation of 3D microstructure of cast Al-Si base alloy containing Si platelets, Fe-rich intermetallics, primary Si particles and pores which are displayed in different colors. Figure 6.3 shows the corresponding 2D serial sections. Substantial additional research is required to develop a general, efficient, and flexible methodology for simulations of realistic multi-phase 3D microstructures of complex engineering alloys.

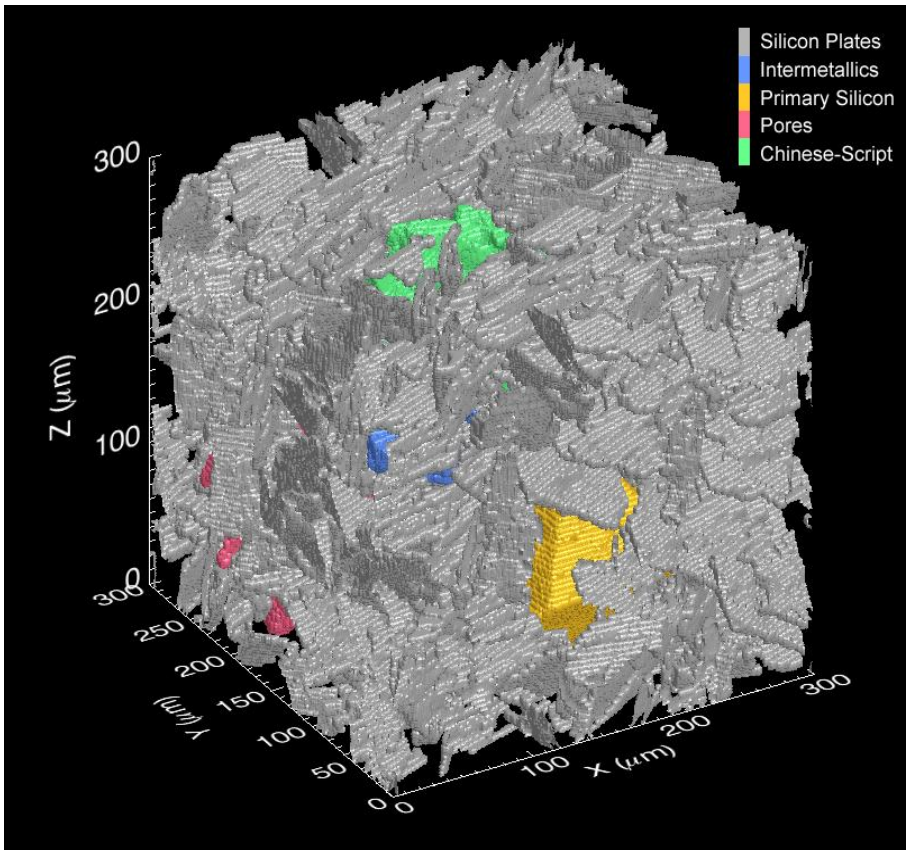


Figure 6.1: Computer simulated multiphase 3D microstructure of AL-Si base alloy

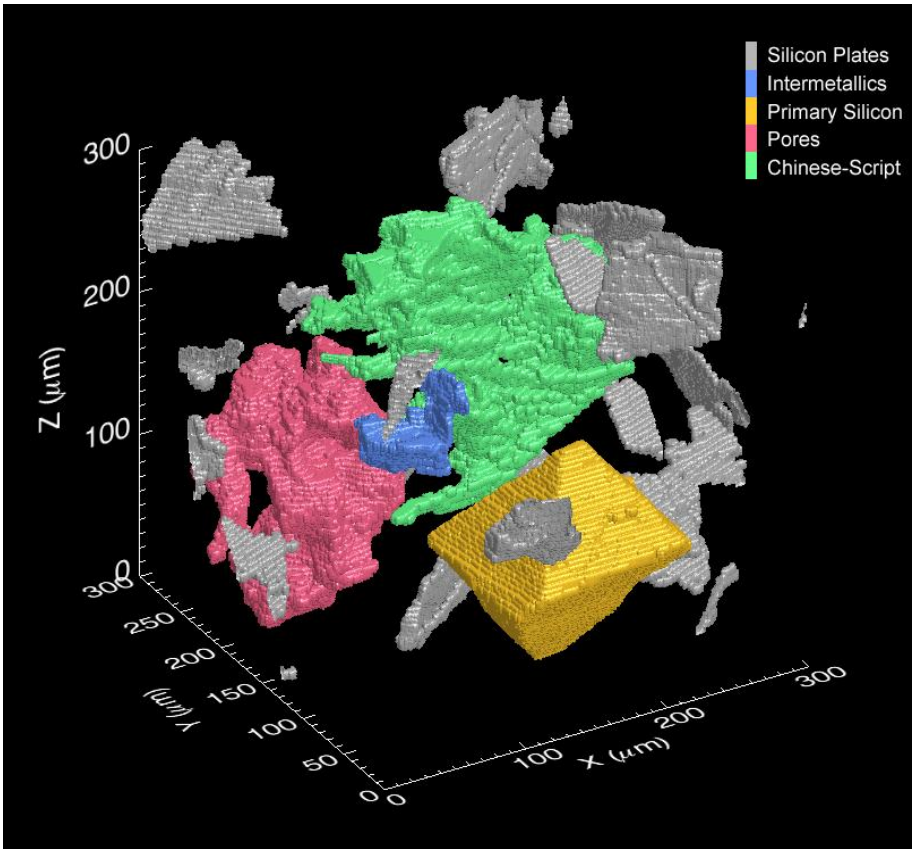
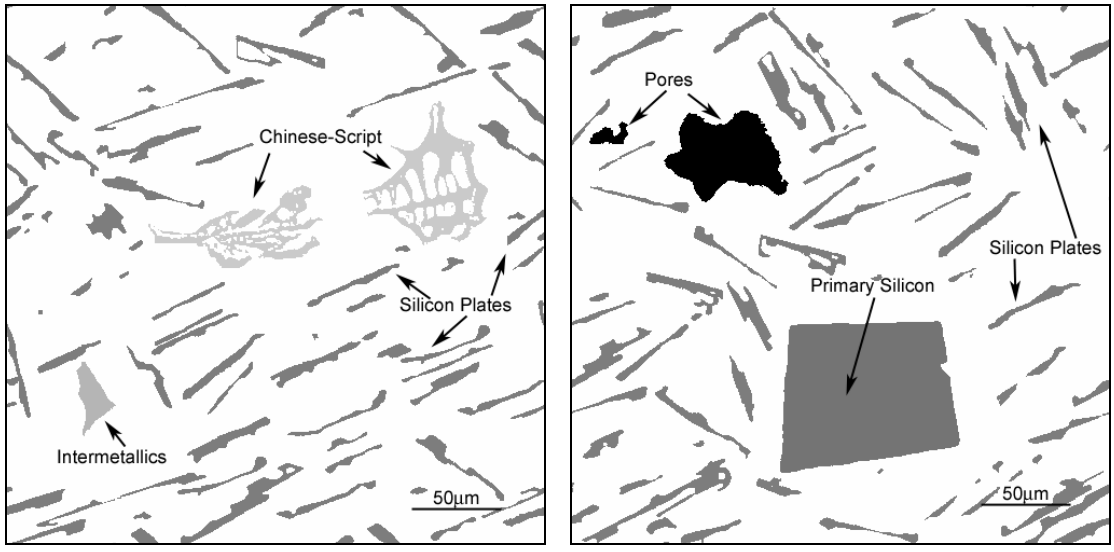


Figure 6.2: The same simulated multiphase 3D microstructure of AL-Si base alloy as Figure 6.1, but with most of the silicon platelets removed to reveal the other phases



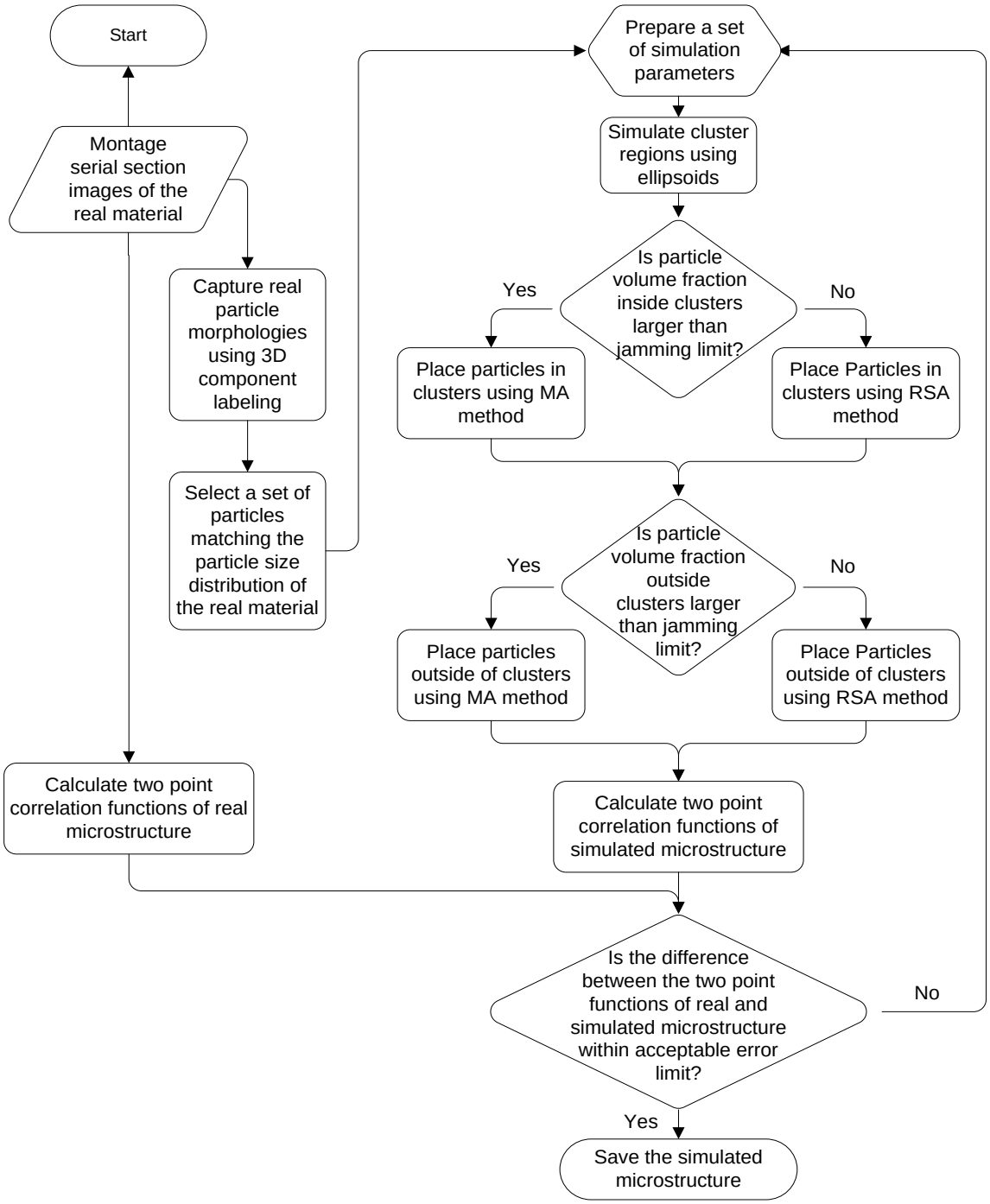
(a) (b)

Figure 6.3: Serial sections of the 3D microstructure in Figure 6.1 through a region containing (a) Chinese script and blocky intermetallic particles and Si platelets; (b) pores, a coarse polyhedral primary Si and Si platelets.

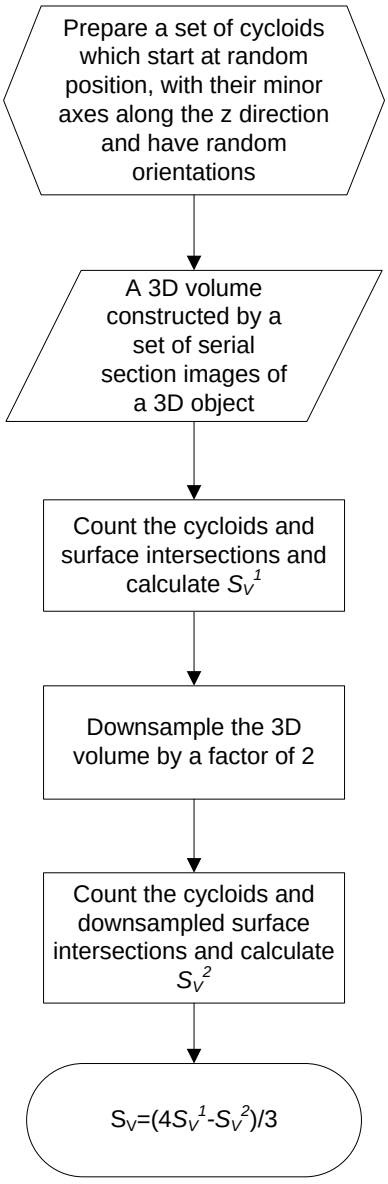
APPENDIX A

FLOWCHART

A.1 Microstructure Simulation



A.2 Dual-Scale Virtual Cycloids



APPENDIX B

SOURCE CODE

```
/*********************************************************************
The source code contained here can be used, copied, modified, merged,
published, and/or have copies distributed for academic or research
purposes only without restriction under the following conditions:

1. The above header and this permission notice shall be included in all
copies or substantial portions of the code.

2. The code is provided "as is", without warranty of any kind, express
or implied, including but not limited to the warranties of
merchantability, fitness for a particular purpose and non-infringement.
In no event shall the author(s) be liable for any claim, damages or
liability, whether in an action of contract, tort or otherwise, arising
from, out of or in connection with this program.

CxImage library and its demo program have been utilized to load, save,
display, and transform images.
CxImage : Copyright (C) 2001 - 2008, Davide Pizzolato
home page: http://www.xdp.it

*********************************************************************/
```

B.1 Two-point Correlation Function

```
void CImageMaxDoc::OnDllTwopoint()
{
 if (image==0) return;
 if (!image->IsValid()) return;
 Dlg2points dlg;
 if (dlg.DoModal()==IDOK)
 {
   m_fp[0]=(void *)(int)dlg.m_2pointsR;
   m_fp[1]=(void *)(int)dlg.m_p45;
   m_fp[2]=(void *)(int)dlg.m_Jump;
   m_fp[3]=(void *)(int)dlg.m_UseFrame;
   m_fp[4]=(void *)(int)dlg.m_FrameX;
   m_fp[5]=(void *)(int)dlg.m_FrameY;
   m_fp[6]=(void *)(int)dlg.m_StartX;
   m_fp[7]=(void *)(int)dlg.m_StartY;
   m_MenuCommand=ID_DLL_TWOPOINT;
   hThread=(HANDLE)_beginthread(RunCxImageThread,0,this);
 }
}

void /*unsigned long _stdcall*/ RunCxImageThread(void *lpParam)
{
 CImageMaxDoc *pDoc = (CImageMaxDoc *)lpParam;
 if (pDoc==NULL) return;
```

```
 if (pDoc->image==NULL) return;

  //prepare for elaboration
 pDoc->image->SetProgress(0);
 pDoc->image->SetEscape(0);

 pDoc->SubmitUndo();

 // auxilary thread for progress bar
 pDoc->hProgress = (HANDLE)_beginthread(RunProgressThread,0,pDoc);

 pDoc->Stopwatch(0);

 switch (pDoc->m_MenuCommand)
 {
 case ID_DLL_TWOPOINT:
   {
     CFileDialog
dlgFile(FALSE,"txt","output_2points.txt",OFN_OVERWRITEPROMPT,"Data
Files (*.txt;*.dat)|*.txt; *.dat|All Files (*.*)|*.*||");
     dlgFile.m_ofn.lpstrTitle="Save the 2point function results as: ";

     if (dlgFile.DoModal() == IDOK)
     {
       DWORD x=pDoc->image->GetWidth();
       DWORD y=pDoc->image->GetHeight();

       CString fn = dlgFile.GetPathName();
       char *filename =  (char*)(const char*)fn; //dont change filename

       int m = (int)pDoc->m_fp[0];
       int P45=(int)pDoc->m_fp[1];
       int Jump=(int)pDoc->m_fp[2];
       int useFrame=(int)pDoc->m_fp[3];
       DWORD frameX=(DWORD)pDoc->m_fp[4];
       DWORD frameY=(DWORD)pDoc->m_fp[5];
       int startX=(int)pDoc->m_fp[6];
       int startY=(int)pDoc->m_fp[7];
       if (useFrame == 1)
       {
         if ((frameX>x/2) || (frameY>y/2))
         {
           AfxMessageBox( "Frame is too big!");
           break;
         }
         if ((startX+frameX>x) || (startY+frameY>y))
         {
           AfxMessageBox( "Starting point causes frame outside of the
image!");
           break;
         }
         P45=0;
         m=(frameX>frameY)?frameX:frameY;
         m--;

       }
       BYTE* im=new BYTE[x*y] ;
```

```
        pDoc->image->GetGrayHead(im);
        //m++;

        double* px=new double[4*(m+1)];
              double* py=new double[4*(m+1)];
        double* p45=new double[4*(m+1)];

        int n = fnMy2points(im, m, int(x), int(y), px, py, p45, P45,
Jump, useFrame, frameX, frameY, startX, startY);

        FILE *fp=fopen (filename, "w");
        //ofstream fp( filename );

        int pd, r;
        fprintf(fp, "output x: \nr");
                for (pd=0; pd<4; pd++)
          fprintf(fp,"\tP%i%i",div(pd,2).quot,div(pd,2).rem);


        for (r=0; r<=m; r+=SKIP)
        {
          fprintf(fp, "\n%d", r);
          for (pd=0; pd<4; pd++)
            fprintf(fp, "\t%f", pmx(pd,4*r));
        }

        fprintf(fp, "\n\noutput y: \nr");
        for (pd=0; pd<4; pd++)
          fprintf(fp,"\tP%i%i",div(pd,2).quot,div(pd,2).rem);

        for (r=0; r<=m; r+=SKIP)
        {
          fprintf(fp, "\n%d", r);
          for (pd=0; pd<4; pd++)
            fprintf(fp, "\t%f", pmy(pd,4*r));
        }
        if (P45==1)
        {
          fprintf(fp, "\n\noutput 45: \nr");
          for (pd=0; pd<4; pd++)
            fprintf(fp,"\tP%i%i",div(pd,2).quot,div(pd,2).rem);

          int r1;
          for (r=0; r<=m; r+=SKIP)
          {
            r1=int(1.4142136*r);
            if (r1<=m)
            {
              fprintf(fp, "\n%d", r1);
              for (pd=0; pd<4; pd++)
                fprintf(fp, "\t%f", pm45(pd,4*r1));
            }
          }
        }
        fclose(fp);

        delete [] im;
```

```
      delete [] px;
      delete [] py;
      delete [] p45;
    }
    break;
  }
 }
 pDoc->Stopwatch(1);

 pDoc->image->SetProgress(100);

 pDoc->hThread=0;
 _endthread();
 return ;
}

#define SKIP ((r<100)?(1):(10)) // skip r
#define SKIP2 ((r<500)?(2):(8)) // skip r
#define pmx(x,y) (*(px+(y+x)))
#define pmy(x,y) (*(py+(y+x)))
#define pm45(x,y) (*(p45+(y+x)))
#define pm3p(x,y) (*(p3p+((y)+x)))
#define Im_in2(x,y) (*(buf_in1+y+x)) //to speed up
#define JUMP 5

//particle is white color (255).
//buf_in1: input image
//m: 2point test line length
//x_s, y_x: image size x, y
int fnMy2points(BYTE *buf_in1, int m, int x_s, int y_s,
       double* px, double* py, double* p45, int P45, int JUMP,
       int useFrame, int frameX, int frameY, int startX, int startY
       )
{
 if (useFrame==1)
 {
   P45=0;
   m=(frameX>frameY)?frameX:frameY;
   m--;
 }
 else
 {
   startX=0;
   startY=0;
   frameX=x_s;
   frameY=y_s;
 }
 double* q=new double[m+1];
 int i;

 int pa,pb,pd;
 int xa,xb,ya,yaa,yb1,yb2,r,rr;

 for (i=0; i<4*(m+1); i++)
   *(px+i)=*(py+i)=0;

 yb1=(y_s-1)*x_s-1;
```

```
 for (r = 0; r<=m; r+=SKIP)
 {
   rr=4*r;
   nMy2points= int (56*r/(m+1)+10);
   for (ya=startY+1; ya<=startY+frameY; ya=ya+JUMP)//all A point in
frame
   {
     yaa=(ya-1)*x_s-1;

     yb2=(ya-1+r)*x_s-1;

     for (xa=startX+1; xa<=startX+frameX; xa=xa+JUMP)
     {
       pa=(Im_in2(xa,yaa)!=255);
       /////////x/////////////
       xb=xa+r;
       if (xb<=x_s)
       {
         pb=(Im_in2(xb,yaa)!=255);
         pmx(pa+pa+pb,rr)++;
       }
       //////////y/////////////
       if (yb2<=yb1)
       {
         pb=(Im_in2(xa,yb2)!=255);
         pmy(pa+pa+pb,rr)++;
       }
     }
   }
 }

 for (r=0; r<=m; r+=SKIP)
   q[r]=0;

 for (r=0; r<=m; r+=SKIP)
 {
   rr=4*r;
   for (pd=0; pd<4; pd++)
     q[r]=q[r]+pmx(pd,rr);
 }

 for (r=0; r<=m; r+=SKIP)
 {
   rr=4*r;
   for (pd=0; pd<4; pd++)
   {
     pmx(pd,rr)=double(pmx(pd,rr))/q[r];
   }
 }
///////////////// x done//////////////
 for (r=0; r<=m; r+=SKIP)
   q[r]=0;

 for (r=0; r<=m; r+=SKIP)
   for (pd=0; pd<4; pd++)
     q[r]=q[r]+pmy(pd,4*r);
```

```
 for (r=0; r<=m; r+=SKIP)
   for (pd=0; pd<4; pd++)
   {
     pmy(pd,4*r)=double(pmy(pd,4*r))/q[r];
   }
//////////////////Y done//////////////////////
 if (P45==1)
 {
   int r1;
   for (i=0; i<4*(m+1); i++)
     *(p45+i)=0;

   for (r = 0; r<=m; r+=SKIP)
   {
     r1=int(1.4142136*r);
     if (r1<=m)
     {
       rr=4*r1;
       nMy2points= int (33*r/(m+1)+66);
       for (ya=r+1; ya<=y_s-r; ya=ya+JUMP)//all A point in frame
       {
         yaa=(ya-1)*x_s-1;
         yb1=(ya-1-r)*x_s-1;
         yb2=(ya-1+r)*x_s-1;
         for (xa=1; xa<=x_s-r; xa=xa+JUMP)
         {
           pa=(Im_in2(xa,yaa)!=255);
           xb=xa+r;
           pb=(Im_in2(xb,yb2)!=255);
           pm45(pa+pa+pb,rr)++;
           pb=(Im_in2(xb,yb1)!=255);
           pm45(pa+pa+pb,rr)++;
         }
       }
     }
   }

   for (r=0; r<=m; r++)
     q[r]=0;

   for (r=0; r<=m; r++)
     for (pd=0; pd<4; pd++)
       q[r]=q[r]+pm45(pd,4*r);

   for (r=0; r<=m; r+=SKIP)
     for (pd=0; pd<4; pd++)
     {
       r1=int(1.4142136*r);
       if (r1<=m)
         pm45(pd,4*r1)=double(pm45(pd,4*r1))/q[r1];
     }
   }
//////////////////////45 done//////////////////////////
 delete [] q;
 return 42;
}
```

B.2 Lineal Path Probability Function

```
void CImageMaxDoc::OnDllLinealpath()
{
 if (image==0) return;
 if (!image->IsValid()) return;
 Dlg2points dlg;
 if (dlg.DoModal()==IDOK)
 {
   m_fp[0]=(void *)(int)dlg.m_2pointsR;
   m_fp[1]=(void *)(int)dlg.m_p45;
   m_fp[2]=(void *)(int)dlg.m_Jump;
   m_fp[3]=(void *)(int)dlg.m_UseFrame;
   m_fp[4]=(void *)(int)dlg.m_FrameX;
   m_fp[5]=(void *)(int)dlg.m_FrameY;
   m_fp[6]=(void *)(int)dlg.m_StartX;
   m_fp[7]=(void *)(int)dlg.m_StartY;
   m_MenuCommand=ID_DLL_LINEALPATH;
   hThread=(HANDLE)_beginthread(RunCxImageThread,0,this);
 }
}

void /*unsigned long _stdcall*/ RunCxImageThread(void *lpParam)
{
 CImageMaxDoc *pDoc = (CImageMaxDoc *)lpParam;
 if (pDoc==NULL) return;
 if (pDoc->image==NULL) return;

  //prepare for elaboration
 pDoc->image->SetProgress(0);
 pDoc->image->SetEscape(0);

 pDoc->SubmitUndo();

 // auxilary thread for progress bar
 pDoc->hProgress = (HANDLE)_beginthread(RunProgressThread,0,pDoc);

 pDoc->Stopwatch(0);

 switch (pDoc->m_MenuCommand)
 {
 case ID_DLL_LINEALPATH:
   {

     CFileDialog
dlgFile(FALSE,"txt","output_LinealPath.txt",OFN_OVERWRITEPROMPT,"Data
Files (*.txt;*.dat)|*.txt; *.dat|All Files (*.*)|*.*||");
     dlgFile.m_ofn.lpstrTitle="Save the Lineal Path function results as:
";

     if (dlgFile.DoModal() == IDOK)
     {

       DWORD x=pDoc->image->GetWidth();
       DWORD y=pDoc->image->GetHeight();
```

```
        CString fn = dlgFile.GetPathName();
        char *filename =  (char*)(const char*)fn; //dont change filename

        int m = (int)pDoc->m_fp[0];
        int P45=(int)pDoc->m_fp[1];
        int Jump=(int)pDoc->m_fp[2];
        int useFrame=(int)pDoc->m_fp[3];
        DWORD frameX=(DWORD)pDoc->m_fp[4];
        DWORD frameY=(DWORD)pDoc->m_fp[5];
        int startX=(int)pDoc->m_fp[6];
        int startY=(int)pDoc->m_fp[7];
        if (useFrame == 1)
        {
          if ((frameX>x/2) || (frameY>y/2))
          {
            AfxMessageBox( "Frame is too big!");
            break;
          }
          if ((startX+frameX>x) || (startY+frameY>y))
          {
            AfxMessageBox( "Starting point causes frame outside of the
image!");
            break;
          }
          P45=0;
          m=(frameX>frameY)?frameX:frameY;
          m--;
        }

        BYTE* im=new BYTE[x*y] ;
        pDoc->image->GetGrayHead(im);

        double* px=new double[4*(m+1)];
              double* py=new double[4*(m+1)];
        double* p45=new double[4*(m+1)];

        int n = fnLinealPath(im, m, int(x), int(y), px, py, p45, P45,
Jump, useFrame, frameX, frameY, startX, startY);

        FILE *fp=fopen (filename, "w");

        int pd, r;
        fprintf(fp, "output x: \nr");
        for (pd=0; pd<4; pd+=3)
          fprintf(fp,"\tP%i%i",div(pd,2).quot,div(pd,2).rem);

        for (r=0; r<=m; r++)
        {
          fprintf(fp, "\n%d", r);
          for (pd=0; pd<4; pd+=3)
            fprintf(fp, "\t%f", pmx(pd,4*r));
        }

        fprintf(fp, "\n\noutput y: \nr");
        for (pd=0; pd<4; pd+=3)
          fprintf(fp,"\tP%i%i",div(pd,2).quot,div(pd,2).rem);
```

```
        for (r=0; r<=m; r++)
        {
          fprintf(fp, "\n%d", r);
          for (pd=0; pd<4; pd+=3)
            fprintf(fp, "\t%f", pmy(pd,4*r));
        }
        if (P45==1)
        {
          fprintf(fp, "\n\noutput 45: \nr");
          for (pd=0; pd<4; pd++)
            fprintf(fp,"\tP%i%i",div(pd,2).quot,div(pd,2).rem);

          int r1;
          for (r=0; r<=m; r++)
          {
            r1=int(1.4142136*r);
            if (r1<=m)
            {
              fprintf(fp, "\n%d", r1);
              for (pd=0; pd<4; pd++)
                fprintf(fp, "\t%f", pm45(pd,4*r1));
            }
          }
        }
        fclose(fp);

        delete [] im;
        delete [] px;
        delete [] py;
        delete [] p45;
      }
      break;
    }
  }
  pDoc->Stopwatch(1);

  pDoc->image->SetProgress(100);

  pDoc->hThread=0;
  _endthread();
  return ;
}

// particle is white color (255).
//buf_in1: input image
//m: lineal path test line length
//x_s, y_x: image size x, y
fnLinealPath(BYTE *buf_in1, int m, int x_s, int y_s,
               double* px, double* py, double* p45, int P45,int JUMP,
               int useFrame, int frameX, int frameY, int startX, int
startY
               )
{
  if (useFrame==1)
  {
    P45=0;
    m=(frameX>frameY)?frameX:frameY;
```

```
  m--;
}
else
{
  startX=0;
  startY=0;
  frameX=x_s;
  frameY=y_s;
}
double* q=new double[m+1];
int i;

int pa,pb,pd;
int xa,xb,ya,yaa,yb,yb2,r,rr;

for (i=0; i<4*(m+1); i++)
  *(px+i)=*(py+i)=0;

int len_cnt,counter;
yb2=(y_s-1)*x_s-1;
for (ya=startY+1; ya<=startY+frameY; ya=ya+JUMP)//all A point in frame
{
  /////////x/////////////
  nMy2points= int (30*ya/(y_s+1)+10);
  yaa=(ya-1)*x_s-1;
    pa=(Im_in2(1,yaa)!=255);
  len_cnt=1;

  for (xa=startX+2; xa<=startX+frameX; xa=xa+JUMP)
  {
    pb=(Im_in2(xa,yaa)!=255);
    if (pa==pb)
    {
      len_cnt++;
    }
    else
    {
      counter=len_cnt;
      if (len_cnt>m) counter=m+1;
      for (i=0;i<counter;i++)
        pmx(pa+pa+pa,(i<<2))+=len_cnt-i;
      pa=pb;
      len_cnt=1;
    }
 }
  counter=len_cnt;
  if (len_cnt>m) counter=m+1;
  for (i=0;i<counter;i++)
    pmx(pa+pa+pa,(i<<2))+=len_cnt-i;
}

for (r=0; r<=m; r++)
  q[r]=(frameX-r)*frameY;

for (r=0; r<=m; r++)
{
  rr=4*r;
```

```
    for (pd=0; pd<4; pd++)
    {
      pmx(pd,rr)=double(pmx(pd,rr))/q[r];
    }
  }
///////////////// x done///////////////
  for (xa=startX+1; xa<=startX+frameX; xa=xa+JUMP)//all A point in frame
  {
    /////////y/////////////
    nMy2points= int (30*xa/(x_s+1)+40);

    pa=(Im_in2(xa,-1)!=255);
    len_cnt=1;

    for (ya=startY+2; ya<=startY+frameY; ya=ya+JUMP)
    {
      yaa=(ya-1)*x_s-1;
      pb=(Im_in2(xa,yaa)!=255);
      if (pa==pb)
      {
        len_cnt++;
      }
      else
      {
        counter=len_cnt;
        if (len_cnt>m) counter=m+1;
        for (i=0;i<counter;i++)
          pmy(pa+pa+pa,(i<<2))+=len_cnt-i;
        pa=pb;
        len_cnt=1;
      }
    }
    counter=len_cnt;
    if (len_cnt>m) counter=m+1;
    for (i=0;i<counter;i++)
      pmy(pa+pa+pa,(i<<2))+=len_cnt-i;
  }

  for (r=0; r<=m; r++)
    q[r]=(frameY-r)*frameX;

  for (r=0; r<=m; r++)
    for (pd=0; pd<4; pd++)
    {
      pmy(pd,4*r)=double(pmy(pd,4*r))/q[r];
    }
////////////////////Y done/////////////////////
  if (P45==1)
  {
    int r1;

    for (i=0; i<4*(m+1); i++)
      *(p45+i)=0;

    for (ya=1; ya<=y_s; ya=ya+JUMP)//all A point in frame
    {
      nMy2points= int (33*ya/y_s+66);
```

```
      yaa=(ya-1)*x_s-1;

      for (xa=1; xa<=x_s; xa=xa+JUMP)
      {
        pa=(Im_in2(xa,yaa)!=255);

        pb=0;
        yb=yaa;
        for (xb=xa; xb<=x_s; xb++)
        {
          if ((pa!=(Im_in2(xb,yb)!=255)) || (int((xb-xa)*1.4142136)>m))
            break;
          pm45(pa+pa+pb,(int(1.4142136*(xb-xa))<<2))++;

          yb+=x_s;
          if (yb>yb2)
            break;
        }

        pb=1;
        yb=yaa;
        for (xb=xb; xb<=x_s; xb++)
        {
          if (int((xb-xa)*1.4142136)>m)
            break;
          pm45(pa+pa+pb,(int(1.4142136*(xb-xa))<<2))++;
          yb+=x_s;
          if (yb>yb2)
            break;
        }
      }
    }

    for (r=0; r<=m; r++)
      q[r]=0;

    for (r=0; r<=m; r++)
      for (pd=0; pd<4; pd++)
        q[r]=q[r]+pm45(pd,4*r);

    for (r=0; r<=m; r++)
      for (pd=0; pd<4; pd++)
      {
        r1=int(1.4142136*r);
        if (r1<=m)
          pm45(pd,4*r1)=double(pm45(pd,4*r1))/q[r1];
      }
  }
//////////////////////45 done//////////////////////////
  delete [] q;
  return 42;
}
```

B.3 2D Contour Tracing

```
void CImageMaxDoc::OnDllContour()
{
 CFileDialog
 dlgFile(FALSE,"txt","output_contour.txt",OFN_OVERWRITEPROMPT,"Data
 Files (*.txt;*.dat)|*.txt; *.dat|All Files (*.*)|*.*||");

 dlgFile.m_ofn.lpstrTitle="Save the contour data points as: ";

 if (dlgFile.DoModal() == IDOK)
 {
   CImageMaxDoc *NewDoc=(CImageMaxDoc*)((CImageMaxApp*)AfxGetApp())-
>demoTemplate->OpenDocumentFile(NULL);
   CxImage *newout = new CxImage(*image);
   NewDoc->image = newout;

   NewDoc->m_fFilename = dlgFile.GetPathName();
   NewDoc->m_MenuCommand=ID_DLL_CONTOUR;
   NewDoc->hThread=(HANDLE)_beginthread(RunCxImageThread,0,NewDoc);

   CString s;
   s.Format("Contour of %s",GetTitle());
   NewDoc->SetTitle(s);
 }
}

void /*unsigned long _stdcall*/ RunCxImageThread(void *lpParam)
{
 CImageMaxDoc *pDoc = (CImageMaxDoc *)lpParam;
 if (pDoc==NULL) return;
 if (pDoc->image==NULL) return;

  //prepare for elaboration
 pDoc->image->SetProgress(0);
 pDoc->image->SetEscape(0);

 pDoc->SubmitUndo();

 // auxilary thread for progress bar
 pDoc->hProgress = (HANDLE)_beginthread(RunProgressThread,0,pDoc);

 pDoc->Stopwatch(0);

 switch (pDoc->m_MenuCommand)
 {
 case ID_DLL_CONTOUR:
   {
     char *filename =  (char*)(const char*)pDoc->m_fFilename; //dont
change filename

     DWORD x=pDoc->image->GetWidth();
     DWORD y=pDoc->image->GetHeight();

     BYTE* im=new BYTE[x*y] ;
     BYTE* imout= new BYTE[x*y] ;
```

```
      for(int i= 0; i< int (x*y); i++)
        imout[i]=0;

      pDoc->image->GetGrayHead(im);

//getcontour
      int n = fnContour(im, imout, int(x), int(y), filename);

      pDoc->image->SetGrayHead(imout);

      delete [] im;
      delete [] imout;
      break;
    }
  }
  pDoc->Stopwatch(1);

  pDoc->image->SetProgress(100);

  pDoc->hThread=0;
  _endthread();
  return ;
}

# define Im_in1(x,y) (*(buf_in1+((y-1)*x_s+x-1)))
# define Im_large_in(x,y) (*(large_in+((y-1)*(x_s+2)+x-1)))
# define Im_out(x,y) *(buf_out+((y-1)*x_s+x-1))

int fnContour(BYTE *buf_in1, BYTE *buf_out, int x_s, int y_s, char
*filename)
{
  BYTE *large_in; // this is a image of size x_s+2 x y_s+2 with the
extra border made up of zeros

  large_in = new BYTE[(x_s+2)*(y_s+2)];

    if(large_in == NULL ) return (666);

  int i,j;

  for (i=0; i< (x_s+2)*(y_s+2); i ++)
    *(large_in+i)=0;

  FILE *fp;

  struct chain
  {
    int x, y;
    struct chain *nextchain;
  };

     struct object1
  {
    struct chain  *chainhead;
    struct object1 *nextobject;
  };
```

```
 struct object1 *objhead, *objtemp, *objpass;
 struct chain *chaintemp, *chainpass;
 objpass=objhead=NULL;
 int offsetx[8]={1,1,0,-1,-1,-1,0,1};
 int offsety[8]={0,-1,-1,-1,0,1,1,1};

// contour
 int prev_grey_val;
 int no_obj=0,m=0,l,tcode[7],is_obj=1,n=0;
 int xc, yc, xi, yi, ck,xobj,yobj;

 int tcode0[8]={7,7,1,1,3,3,5,5};
 int k_out[8]={3,4,5,6,7,0,1,2};
 int k_inner[8]={7,0,1,2,3,4,5,6};
 int pcode;

//initial the value of large_in
 for (j=1; j<=y_s;j++)
 {
   int jj = j*(x_s+2);
   int jj2 = (j-1)*x_s -1;
   nContour= long (10*j/y_s);
   for (i=1; i<=x_s;i++)
   *(large_in+jj+i) = *(buf_in1 +jj2 +i);
 }


 for (j=2; j<=y_s+1;j++)
 {
   nContour= long (85*j/y_s+10);
   for (i=2; i<=x_s+1;i++)
   {
     prev_grey_val = Im_large_in(i-1,j);

     if ((Im_large_in(i,j)==255 && prev_grey_val==0) ||
(Im_large_in(i,j)==255 && prev_grey_val==2) )
     {// This is outer-loop
       Im_out(i-1,j-1)=255;

       chainpass=new chain; //initial
       xobj=xc = chainpass->x=i;
       yobj=yc = chainpass->y=j;
       chainpass->nextchain=NULL;
         objtemp=new object1;
       objtemp->nextobject=NULL;
       objtemp->chainhead=chainpass;

       if(objhead==NULL) objhead=objtemp;
       else objpass->nextobject=objtemp;
       objpass=objtemp;

       pcode = 7; // initial value at the start of the outer-loop

       Im_large_in(i-1,j)=2;

       l=0;
```

```
while (l<=6)
{
  if (l==0)
    tcode[l]=tcode0[pcode]; // tcode
  else
    tcode[l]=(tcode[l-1]+1)%8;

  xi = xc + offsetx[tcode[l]];
    yi = yc + offsety[tcode[l]];

  if (Im_large_in(xi,yi)==0 || Im_large_in(xi,yi)==2)
  {
    Im_large_in(xi,yi)=2;
    l++;
  }
  else
  {
    Im_out(xi-1,yi-1)=255;
    Im_large_in(xi,yi)=3;
    if (xi==xobj && yi==yobj)
    {
      ck=(tcode[l]+4)%8;
      n=k_out[ck]+1;
      xc=xi;
      yc=yi;

     if (n>6)
      {
        l=n;
      }//new

      while (n<=6)
      {
        tcode[l]=(ck+1)%8; // tcode
        xi = xc + offsetx[tcode[l]];
        yi = yc + offsety[tcode[l]];

        if (Im_large_in(xi,yi)==0 || Im_large_in(xi,yi)==2)
        {
          Im_large_in(xi,yi)=2;
          n++;
          l=n;
        }
        else
        {
          Im_large_in(xi,yi)=3;

          chaintemp=new chain;
          chaintemp->x=xi;
          chaintemp->y=yi;
          chaintemp->nextchain=NULL;

          chainpass->nextchain=chaintemp;
          chainpass=chaintemp;

          xc=xi;
```

```
                    yc=yi;
                    pcode=tcode[l];
                    l=0;
                    n=7;
                  }
                }//end of n while
              }
              else
              {
               chaintemp=new chain;
               chaintemp->x=xi;
               chaintemp->y=yi;
               chaintemp->nextchain=NULL;
               chainpass->nextchain=chaintemp;
               chainpass=chaintemp;

               xc=xi;
               yc=yi;
               pcode=tcode[l];
               l=0;
              }
          }
         }//end of while
       no_obj++;
       m=0;
    } //end of the outer-loop if

// output
 fp=fopen (filename, "w");

 fprintf(fp, "Total_Particle_Number: %d", no_obj);
 m=1;
 objtemp=objhead;
 while (objtemp != NULL)
 {
   chaintemp=objtemp->chainhead;
   while (chaintemp != NULL)
   {
     fprintf(fp, "\n%d %d", chaintemp->x, chaintemp->y);
     chainpass=chaintemp->nextchain;
     delete chaintemp;
     chaintemp=chainpass;
   }

   fprintf(fp, "\n-100 %d", m++);

   objpass=objtemp->nextobject;
   delete objtemp;
   objtemp=objpass;
 }

 fclose(fp);
//output done
   delete [] large_in;
 return no_obj;
}
```

B.4 2D Cluster Simulation

```
//Ellipsis simulation
struct list_elem_2D
{
 double x,y,a,b,rot,dFin;
 list_elem_2D* next;
};

struct ellipse
{
  double x;
  double y;
  double a;
  double b;
  double rot;
  double dFin;

};

class list_2D
{
 list_elem_2D* h;
public:
 list_2D() {h=0;};
 ~list_2D() {release();}
 void add(double x, double y, double a, double b, double rot, double
dFin);
 void del() { list_elem_2D* temp=h;
        h=h->next;
        delete temp;}
 list_elem_2D* first() {return (h); }
 bool intersect(double x, double y, double a, double b, double rot);
 bool inside(double x, double y);
 void pr_list_2D(FILE *fp,double global_x, double global_y,struct
ellipse *ellp, int* i);
 void release();
};

void list_2D::add(double x, double y, double a, double b , double rot,
double dFin)
{
 list_elem_2D* temp = new list_elem_2D;
 temp->next=h;
 temp->x = x;
 temp->y = y;
 temp->a = a;
 temp->b = b;
 temp->rot= rot;
 temp->dFin = dFin;
 h=temp;
}

bool list_2D::intersect(double x, double y, double a, double b, double
rot)
{
```

```
 list_elem_2D* temp=h;
 bool not_intersect=true;
 double dx, dy, dia, p_x, p_y, p_xrot, p_yrot, c, tempc;
 int alpha;
 while ( (temp!=0) && not_intersect)
 {
   dx = temp->x-x;
   dy = temp->y-y;
   dia = temp->a+a;
   not_intersect = ( (dx*dx + dy*dy) > dia*dia );
   // assume the major axis of ellipse is along the x axis;
   // thus focus of the ellipse is (c,0) and (-c,0)
   // need check both sides. cos ellipse can be small enough to fit in
larger one.
   if (!not_intersect)
   {
     not_intersect= true;

     c = sqrt(a*a-b*b);
     alpha=0;
     while ((alpha<=360)&& not_intersect)
     {
       p_x =temp->a* cos(alpha*Pi/180);
       p_y =temp->b* sin(alpha*Pi/180);
         p_xrot = dx + p_x*cos(temp->rot) - p_y*sin(temp->rot);
                p_yrot = dy + p_x*sin(temp->rot) + p_y*cos(temp->rot);
       p_x= p_xrot*cos(rot) + p_yrot*sin(rot);
       p_y= -p_xrot*sin(rot) + p_yrot*cos(rot);
       not_intersect =( (sqrt((p_x-c)*(p_x-c)+p_y*p_y)
+sqrt(p_y*p_y+(p_x+c)*(p_x+c)))  > 2*a);
       alpha+=1;

     }

     if (not_intersect)//check if object is totally inside of the temp
     {
       tempc = sqrt(temp->a*temp->a-temp->b*temp->b);
         p_x =a* cos(alpha*Pi/180);
       p_y =b* sin(alpha*Pi/180);
           p_xrot = -dx + p_x*cos(rot) - p_y*sin(rot);
                p_yrot = -dy + p_x*sin(rot) + p_y*cos(rot);
       p_x= p_xrot*cos(temp->rot) + p_yrot*sin(temp->rot);
       p_y= -p_xrot*sin(temp->rot) + p_yrot*cos(temp->rot);
         not_intersect =( (sqrt((p_x-tempc)*(p_x-tempc)+p_y*p_y)
+sqrt(p_y*p_y+(p_x+tempc)*(p_x+tempc)))  > 2*temp->a);
     }
   }
   temp=temp->next ;
 }
 return (!not_intersect);
}

void list_2D::pr_list_2D(FILE *fp,double global_x, double
global_y,struct ellipse* ellps, int* i)
{
 list_elem_2D* temp=h;
 while (temp!=0)
```

```
 {
   fprintf(fp,"%f\t%f\t%f\t%f\t%f\t%f\n",global_x+temp-
>x,global_y+temp->y,temp->a, temp->b, temp->rot, temp->dFin);
   ellps[*i].x=global_x+temp->x;
   ellps[*i].y=global_y+temp->y;
   ellps[*i].a=temp->a;
   ellps[*i].b=temp->b;
   ellps[*i].rot=temp->rot;
   ellps[*i].dFin=temp->dFin;
   *i=*i+1;
   temp=temp->next ;
 }
 release();
}

void list_2D::release()
{
 while (h!=0)
   del();
}

class matrix_2D
{
 list_2D** p;
 long s1, s2;
public:
 long ub1, ub2;
 matrix_2D(long d1, long d2);
// ~matrix_2D();
 void del();
 list_2D& element(long i, long j);
};

matrix_2D::matrix_2D(long d1, long d2)
{
 s1=d1;
 s2=d2;
 p = new list_2D*[s1];
 for (long i =0; i<s1;i++)
   p[i] = new list_2D[s2];
 ub1=s1-1;
 ub2=s2-1;
}

void matrix_2D::del()
{
  for (long i =0; i<=ub1; i++)
   delete [] p[i];
 delete [] p;
}


list_2D& matrix_2D::element(long i, long j)
{
 long x=i,y=j;
 if (i<0) x=ub1+1+i;
 if (i>ub1) x=i-ub1-1;
```

```
 if (j<0) y=ub2+1+j;
 if (j>ub2) y=j-ub2-1;
 return(p[x][y]);
}

int D2(struct ellipse *ellp, long Lx, long Ly, char *ellipsefilename,
double Aa, double ellpda, double ellpdb)
 {
 char line[150];
 long SimID;
 long Ntot;
 double Area, Na, rotat;
 char pattenhead[] = "SIZEA  SIZEB NUMBER  DENSITY ORIENTATION";

 struct elipse_size
 {
   double elipa,elipb, dFin,rotat;
 };
 double sizemax=0;
 elipse_size* size_array;
 long i,j,w; //loop variables

 Area=Lx*Ly;

 FILE *stream = fopen( ellipsefilename , "rt");

 fgets( line, 150, stream );
 fgets( line, 150, stream );
 fgets( line, 150, stream );
 if (strstr( line, pattenhead ) == NULL)
 {
   fclose( stream );
   return -3;
 }

 double siza, sizb, freq, freq_sum=0, narea=0, dFin;

 while (!feof(stream))
 {
   fscanf( stream, "%lf %lf %lf %lf %lf",&siza, &sizb, &freq, &dFin,
&rotat);
   freq_sum += freq+int(Aa);// Aa for batch mode adjustment
   narea=narea+(siza+ellpda)*(sizb+ellpdb)*(freq+int(Aa))/4*Pi;
   if (siza>sizemax)
     sizemax = siza;
 }

 fclose( stream );

 Ntot = int(freq_sum);

    size_array = new elipse_size[Ntot+1];

 if (Ntot == 0 || Ntot > ELLIPENOLIMIT || narea/Area > 0.6 )
 {
     delete [] size_array;
```

```
    return 0;
}

stream = fopen( ellipsefilename, "rt");
fgets( line, 150, stream );
fgets( line, 150, stream );
fgets( line, 150, stream );
i=0;

while (!feof(stream))
{
  fscanf( stream, "%lf %lf %lf %lf %lf",&siza, &sizb, &freq, &dFin,
&rotat);

  for (j=0;j<(freq+int(Aa));j++)
  {
           size_array[i].rotat=rotat;
    size_array[i].dFin=dFin;
    size_array[i].elipa=siza+ellpda;// for batch mode, changing the
ellpise's a
    size_array[i++].elipb=sizb+ellpdb;// for batch mode, changing the
ellpise's b
  }
}
fclose( stream );

//randomize the size distribution
SimID = (unsigned)time( NULL );
//SimID = 1003;
seedMT(SimID);
double temp_sizea, temp_sizeb, temp_dFin, temp_rotat;
long ran_n;
for ( i =0;i<Ntot;i++)
{
  temp_sizea=size_array[i].elipa;
  temp_sizeb=size_array[i].elipb;
  temp_dFin=size_array[i].dFin;
  temp_rotat=size_array[i].rotat;
  ran_n =long(Ntot*random53());
  size_array[i]=size_array[ran_n];
  size_array[ran_n].elipa=temp_sizea;
  size_array[ran_n].elipb=temp_sizeb;
  size_array[ran_n].dFin=temp_dFin;
  size_array[ran_n].rotat=temp_rotat;
}
//"randomize the size distribution"Completed

Na=Ntot/Area;

double cell_size_x, cell_size_y;
cell_size_x = cell_size_y = sizemax*3;

long lx,ly; // lx, ly are the no. of cells in x and y directions
lx= long (Lx/cell_size_x+1);
ly= long (Ly/cell_size_y+1);

if (lx == 0 || ly == 0)
```

```
 {
   delete [] size_array;
   return 0;
 }

 matrix_2D map_2D(lx,ly); // map_2d is the simulation space

 cell_size_x= double(Lx)/lx;
 cell_size_y= double(Ly)/ly;

 // you can seed with any uint32, but the best are odds in 0..(2^32 - 1)
 SimID = (unsigned)time( NULL );
 //SimID = 1003;
 seedMT(SimID);

 double rac,rbc,xc,yc; //current variables
 for ( w=0; w<Ntot; w++)
 {
   bool not_intersect = false;
   long ic,jc;
   rac=size_array[w].elipa/2; //current radius
   rbc=size_array[w].elipb/2;
   rotat=Pi/180*size_array[w].rotat*(random53()*2-1);
   //rotat=0.0;
   while (not_intersect==false)
   {
     not_intersect = true;
     xc=lx*random53(); // this gives a random real number (double)
between 0 and lx
     yc=ly*random53(); // this gives a random real number (double)
between 0 and ly

     ic=long (floor(xc));
     jc=long (floor(yc));

     for ( i=0;(i<=2)&& not_intersect;i++)
       for ( j=0;(j<=2)&& not_intersect;j++)
       not_intersect = !map_2D.element(i+ic-1,j+jc-1).intersect( ( xc-
ic+(1-i) )*cell_size_x,( yc-jc+(1-j) )*cell_size_y,rac, rbc, rotat);
     // this is sending the coordinates of the current point in real
scale wrt  (i+ic-1,j+jc-1) as origin
     // note that (xc-ic,yc-jc) is the coordinate of the point in
apparent scale with the cell (ic,jc) as origin.
     // Thus, ( xc-ic+(1-i),yc-jc+(1-j) ) is the coordinate of the
point in apprant scale with the
     // cell (i+ic-1,j+jc-1) as the origin.
     // Finally the factor *cell_size_x and cell_size_y are used to
transforn the scale to real scale
   }
   map_2D.element(ic,jc).add( (xc-ic)*cell_size_x, (yc-
jc)*cell_size_y,rac,rbc,rotat,size_array[w].dFin);
   // this adds a local
  // coordinate of the cell ic, jc
  // in real scale
 }

 stream = fopen( "output.txt", "w");
```

```
fprintf(stream,"RandomID\tFramesizeX\tFramesizeY\tTotalNumber\tAa\tNa\n
");

fprintf(stream,"%d\t%f\t%f\t%d\t%f\t%3.15f\n",SimID,lx*cell_size_x,ly*c
ell_size_y, Ntot, narea/Area, Na);

fprintf(stream,"CenterX\tCenterY\tSizeA\tSizeB\tOrientation\tInsideDens
ity\n");

 int ellpnumber=1;//for output of ellp

 for (i=0;i<lx;i++)
 {
   for(j=0;j<ly;j++)
   {
       map_2D.element(i,j).pr_list_2D(stream,
i*cell_size_x,j*cell_size_y,ellp,&ellpnumber);
   }
 }
 fclose(stream);
 map_2D.del();
 delete [] size_array;
 return Ntot;
}
```

B.5 2D Microstructure Simulation

```
void CImageMaxApp::OnDllSimupart()
{
 dlgsimupart dlg;
 FILE *fp;
 char pattenhead[] = "FRAMEX FRAMEY  Aa  P45 REPEAT  R2POINT SHAPEFILE";
 char line[150];
 int height, width, P45, repeat, r2point;
 float aa;

 if (dlg.DoModal()==IDOK)
 {
   char* filename1 =  (char*)(const char*)dlg.m_EllipseDisFname;
   if (dlg.m_EllipseDisFname == "")
     {
       AfxMessageBox( "Can't open the data file!\n");
       return;
     }
   else
       if( ( fp=fopen( filename1, "r" )) == NULL )
     {
       AfxMessageBox( "Can't open the data file!\n");
       return;
     }

   fgets( line, 150, fp);
```

```
   if (strstr( line, pattenhead ) == NULL)
   {
     fclose(fp);
     AfxMessageBox( "Error data file format!\n");
     return;
   }

   fscanf(fp, "%d  %d  %f  %d  %d  %d  %s",&width, &height, &aa, &P45,
&repeat, &r2point, line);

   fclose(fp);


   char* filename2 =  (char*)(const char*)line;
   if( ( fp=fopen( filename2, "r" )) == NULL )
   {
       AfxMessageBox( "Can't open the shape data file!\n");
       return;
   }
   fclose(fp);

   CImageMaxDoc *NewDoc=(CImageMaxDoc*)((CImageMaxApp*)AfxGetApp())-
>demoTemplate->OpenDocumentFile(NULL);

   NewDoc->m_fAa=aa;
   NewDoc->m_fp[1]=(void *)(int)height;
   NewDoc->m_fp[0]=(void *)(int)width;
   NewDoc->m_fp[2]=(void *)(int)dlg.m_chkoverlap;
   NewDoc->m_fp[3]=(void *)(int)dlg.m_DrawEllipse;
   NewDoc->m_fp[4]=(void *)(int)dlg.m_Batchsimupart;
   NewDoc->m_fp[5]=(void *)(int)P45;
   NewDoc->m_fp[6]=(void *)(int)repeat;
   NewDoc->m_fp[7]=(void *)(int)r2point;
   NewDoc->m_fFilename = dlg.m_EllipseDisFname;
   NewDoc->m_fFilename2 = line;

   CxImage *newout = new CxImage(width,height,8);
   NewDoc->image = newout;

   NewDoc->m_MenuCommand=ID_DLL_SIMUPART;
   NewDoc->hThread=(HANDLE)_beginthread(RunSimuThread,0,NewDoc);

   CString s;
   s.Format("simulation result");
   NewDoc->SetTitle(s);
   NewDoc->UpdateAllViews(0,WM_USER_NEWIMAGE);
 }
}

void RunSimuThread(void *lpParam)
{
 POSITION posView;
 CView *pView;

 CImageMaxDoc *pDoc = (CImageMaxDoc *)lpParam;

 if (pDoc==NULL) return;
```

```
 if (pDoc->image==NULL) return;

  //prepare for elaboration
 pDoc->image->SetProgress(0);
 pDoc->image->SetEscape(0);

 nSimu2d=0;
 pDoc->Stopwatch(0);

 pDoc->hProgress = (HANDLE)_beginthread(RunSimuProgressThread,0,pDoc);

 switch (pDoc->m_MenuCommand)
 {
   case ID_DLL_SIMUPART:
     {
       int n,i, P45;
       P45=(int)pDoc->m_fp[5];

       char* filename =  (char*)(const char*)pDoc->m_fFilename;
       char* filename2 =  (char*)(const char*)pDoc->m_fFilename2;

       BYTE* imout= new BYTE[((int)pDoc->m_fp[0]+2)*((int)pDoc-
>m_fp[1]+2)] ;

       CSimupart prtcl(filename2);
     //  n = fnSimupartshape(filename2);

       if (prtcl.prtclhead==NULL) //number of particle may change due
to the size limit.
       {
         AfxMessageBox( "NOT ENOUGH RAM TO RUN THIS SIMULATION!");
         delete [] imout;
         break;
       }

       for(i= 0; i< (((int)pDoc->m_fp[0]+2)*((int)pDoc->m_fp[1]+2));
i++)
       imout[i]=0;

       n = fnSimupart(imout,(int)pDoc->m_fp[0],(int)pDoc->m_fp[1],
filename,
         filename2, pDoc->m_fAa, 0.0, (int)pDoc->m_fp[2],
         (int)pDoc->m_fp[3], 0.0, 0.0, 0.0, prtcl.prtclhead);

       if (n==0)
       {
         AfxMessageBox( "Can't Simulate! Please check cluster size
distribution file!\n The Image Size maybe too small to fit the cluster
size distribution!");
         delete [] imout;
         break;
       }

       if (n==-3)
       {
         AfxMessageBox( "Wrong ellipse size file format!");
         delete [] imout;
```

```
          break;
        }

        if (n==2)
        {
          AfxMessageBox( "NOT ENOUGH RAM TO RUN THIS SIMULATION!");
          delete [] imout;
          break;
        }

        pDoc->image->SetGrayHeadfast(imout);

        delete [] imout;
        //delete prtcl;
        break;
      }//end case ID_DLL_SIMUPART

 //pDoc->image->SetProgress(100);
 nSimu2d=100;
 pDoc->Stopwatch(1);

 pDoc->hThread=0;
 _endthread();
 return ;
}

#define ELLIPENOLIMIT 800 //cluster number limit

int fnSimupart(unsigned char *buf_out, int imageWidth, int imageHeight,
char *ellipsefilename,  char *particlefilename, double  vol_frac,
double Aain, int overlap_check, int ellipse_boundary, double ellipse_Aa,
double ellpda,double ellpdb, particle *prtclhead)
{
// calc ellipse info
 int no_ellipse;
 struct ellipse *ellp=new struct ellipse[ELLIPENOLIMIT+1];
 no_ellipse = D2(ellp, imageWidth, imageHeight,
ellipsefilename,ellipse_Aa,ellpda,ellpdb);

 if (no_ellipse==0 || no_ellipse==-3)
 {
   delete [] ellp;
   return no_ellipse;
 }

// simulation, placing particles
 int n =
simulate(ellp,prtclhead,buf_out,no_ellipse,imageWidth,imageHeight,
vol_frac, Aain, overlap_check, ellipse_boundary);

 delete [] ellp;

 if (n==2)
   return n;

 return 42;
}
```

```
//2D simulation
int simulate(struct ellipse *ellp ,particle *prtclhead,unsigned char
*imgdata,int no_of_ellipse, int borderx, int bordery, double vol_frac,
double Aain, int overlap_check, int ellipse_boundary)
{
 int xx,yy;

//particles inside start ////////////
 struct dpoint
 {
   double x,y;
 };

 struct dpoint F1,F2,T;

 struct point *pfillpass;
 particle *prtclpass=prtclhead;

 FILE *ch=fopen("ch3.txt","w");

 int frame_area=borderx*bordery;
 int cp,pixel_count_in,finish=0;
 double d1,d2,randx,randy,area_frac,overall_ar_frac;
 double area_elp,f_length;
 int overall_pixel_count=0;
 bool overlapflag;

 long SimID;
 SimID = (unsigned)time( NULL );
 seedMT(SimID);

 for (cp=1;cp<=no_of_ellipse;cp++)
 {
     area_elp=ellp[cp].a*ellp[cp].b*Pi;
   int lower_limit=int((ellp[cp].dFin+Aain)*area_elp);
   //focus points
   f_length=sqrt(ellp[cp].a*ellp[cp].a-ellp[cp].b*ellp[cp].b);

   F1.x=ellp[cp].x-f_length*cos(ellp[cp].rot);
   F1.y=ellp[cp].y-f_length*sin(ellp[cp].rot);
   F2.x=ellp[cp].x+f_length*cos(ellp[cp].rot);
   F2.y=ellp[cp].y+f_length*sin(ellp[cp].rot);

   int retry_counter=0;
   int pNo_counter=0;

   pixel_count_in=0;

   while (pNo_counter< int(ellp[cp].dFin+Aain))
   {
     randx=random53()*ellp[cp].a*2-ellp[cp].a;
     randy=random53()*ellp[cp].b*2-ellp[cp].b;
     T.x=ellp[cp].x + randx*cos(ellp[cp].rot) - randy*sin(ellp[cp].rot);
     T.y=ellp[cp].y + randx*sin(ellp[cp].rot) + randy*cos(ellp[cp].rot);

     d1= sqrt((T.x-F1.x)*(T.x-F1.x)+(T.y-F1.y)*(T.y-F1.y));
```

```
d2= sqrt((T.x-F2.x)*(T.x-F2.x)+(T.y-F2.y)*(T.y-F2.y));

if ((d1+d2)>2*ellp[cp].a)
{
  retry_counter++;
  if (retry_counter>2000)
    break;
  continue;
}

if (overlap_check==1)
{
  overlapflag = false;
  pfillpass = prtclpass->pfillhead;
  while (pfillpass != NULL)
  {
    xx=int(pfillpass->x-prtclpass->xc+T.x);
    yy=int(pfillpass->y-prtclpass->yc+T.y);

    if (xx<0)
      xx=xx+borderx;
        if (xx>borderx)
      xx=xx-borderx;
          if (yy<0)
      yy=yy+bordery;
        if (yy>bordery)
      yy=yy-bordery;

    if (imgdata[xx+yy*borderx] == 255)
    {
      overlapflag=true;
      break;
    }
    pfillpass = pfillpass->nextpoint;
  }

  if (overlapflag)
  {
    retry_counter++;
    if (retry_counter>2000)
      break;
    continue;
  }
}// end overlap check

pfillpass = prtclpass->pfillhead;
while (pfillpass != NULL)
{
  xx=int(pfillpass->x-prtclpass->xc+T.x);
  yy=int(pfillpass->y-prtclpass->yc+T.y);
  if (xx<0)
    xx=xx+borderx;
      if (xx>=borderx)
    xx=xx-borderx;
        if (yy<0)
    yy=yy+bordery;
    if (yy>=bordery)
```

```
          yy=yy-bordery;

        if (imgdata[xx+yy*borderx]!=255)
        {
          imgdata[xx+yy*borderx]=255;
          pixel_count_in++;
          overall_pixel_count++;
        }
        pfillpass = pfillpass->nextpoint;
      }

      if(prtclpass->nextprtcl!= NULL)
        prtclpass = prtclpass->nextprtcl;
      else
        prtclpass = prtclhead;

      pNo_counter++;

      if (overall_pixel_count>=int(vol_frac*frame_area))
      {
        finish =1;
        break;
      }

    }// end for "while (pixel_count_in<lower_limit)" loop

    overall_ar_frac=double(overall_pixel_count)/frame_area;
    area_frac=pixel_count_in/area_elp;

    fprintf(ch,"area frac = %f retry counter = %d overall = %f
\n",area_frac,retry_counter,overall_ar_frac);
    if (finish==1)
      break;
  }//done inside loop

  fclose(ch);
  double  inside_ar_frac=overall_ar_frac;
//particle outside the clusters//////////////////
  int tot_pixel_out=0;

  while (overall_ar_frac<vol_frac)
  {
    overlapflag=false;

    T.x=borderx*random53();
    T.y=bordery*random53();

    for (cp=1;cp<=no_of_ellipse;cp++)
    {
      f_length=sqrt(ellp[cp].a*ellp[cp].a-ellp[cp].b*ellp[cp].b);

      F1.x=ellp[cp].x-f_length*cos(ellp[cp].rot);
      F1.y=ellp[cp].y-f_length*sin(ellp[cp].rot);
      F2.x=ellp[cp].x+f_length*cos(ellp[cp].rot);
      F2.y=ellp[cp].y+f_length*sin(ellp[cp].rot);

      d1= sqrt((T.x-F1.x)*(T.x-F1.x)+(T.y-F1.y)*(T.y-F1.y));
```

```
  d2= sqrt((T.x-F2.x)*(T.x-F2.x)+(T.y-F2.y)*(T.y-F2.y));

  if ((d1+d2)<2*ellp[cp].a)
  {
    overlapflag=true;
    break;
  }
     }

if (overlapflag)
  continue;

if (overlap_check==1)
{
  pfillpass = prtclpass->pfillhead;
  while (pfillpass != NULL)
  {
    xx=int(pfillpass->x-prtclpass->xc+T.x);
    yy=int(pfillpass->y-prtclpass->yc+T.y);

    if (xx<0)
      xx+=borderx;
    if (xx>=borderx)
      xx-=borderx;
    if (yy<0)
      yy+=bordery;
        if (yy>=bordery)
      yy-=bordery;

    if (imgdata[xx+yy*borderx] == 255)
    {
      overlapflag=true;
      break;
    }
    pfillpass = pfillpass->nextpoint;
  }

  if (overlapflag)
    continue;
}

pfillpass = prtclpass->pfillhead;
while (pfillpass != NULL)
{
  xx=int(pfillpass->x-prtclpass->xc+T.x);
  yy=int(pfillpass->y-prtclpass->yc+T.y);
  if (xx<0)
    xx+=borderx;
  if (xx>=borderx)
    xx-=borderx;
  if (yy<0)
    yy+=bordery;
      if (yy>=bordery)
    yy-=bordery;

  if (imgdata[xx+yy*borderx]!=255)
  {
```

```
        imgdata[xx+yy*borderx]=255;
        tot_pixel_out++;
       }
       pfillpass = pfillpass->nextpoint;
     }

     overall_ar_frac=inside_ar_frac+ double(tot_pixel_out)/frame_area;
     if(prtclpass->nextprtcl != NULL)
       prtclpass = prtclpass->nextprtcl;
     else
       prtclpass = prtclhead;
  }
//outside done/////////////////
  return 42;

}

CSimupart::CSimupart()
{
  prtclhead = NULL;
  return;
}

CSimupart::CSimupart(char *particlefilename)
{
//read particle shape info

  FILE *fp=fopen (particlefilename, "rt");
  char s[30];
  int no_partsize;

  particle *prtclpass, *prtcltemp;
  struct point *pointpass, *pointtemp, *pointhead;

  prtclpass=prtclhead = NULL;

  pointpass = pointhead = NULL;

     int l=1;
  int k=1;
  int i,x,y,perimax=0;
  double perireal=0;

// read prtcl[].perim
  fscanf(fp, "%s %d", s, &no_partsize);
  while (!feof(fp))
  {
    fscanf(fp, "%d %d", &x, &y );
    if (x!=-100)
    {
      pointtemp = new point;
      pointtemp->x = x;
      pointtemp->y = y;
      pointtemp->nextpoint = NULL;
      if(pointhead == NULL)
      {
        perireal++;
```

```
      pointhead = pointtemp;
    }
    else
    {
      if ((pointpass->x != x) && (pointpass->y !=y))
        perireal+=1.414213562373;
      else
        perireal++;
      pointpass->nextpoint=pointtemp;
    }
    pointpass=pointtemp;
    k++;
  }
  else
  {
    prtcltemp = new particle;
    prtcltemp->perim=k-1;
    prtcltemp->perimreal=perireal;
    prtcltemp->area=0;

    prtcltemp->pointhead = pointhead;
    if (perireal>perimax)
      perimax=int(perireal+1);
    prtcltemp->nextprtcl=NULL;

    if(prtclhead==NULL)
      prtclhead=prtcltemp;
    else
      prtclpass->nextprtcl=prtcltemp;
    prtclpass=prtcltemp;

    pointpass = pointhead = NULL;
    k=1;
    perireal=0;
  }
 }
fclose(fp);


// calc praticle shape info
 if (xc_yc_area(prtclhead,perimax) == 2)
 {
   prtclpass=prtclhead;
   while (prtclpass != NULL)
   {
     pointpass = prtclpass->pointhead;
     while (pointpass != NULL)
     {
       pointtemp = pointpass->nextpoint;
       delete pointpass;
       pointpass = pointtemp;
     }
     prtcltemp=prtclpass;
     prtclpass=prtcltemp->nextprtcl;
     delete prtcltemp;
   }
   return;
```

```
 }

 ofstream out("particleshape.txt"); // Write
 out << "no.\tcenterx\tcentery\tperim\tarea\n";
 prtclpass=prtclhead;
 i=1;
 while (prtclpass!=NULL)
 {
   out << i++ << "\t" << prtclpass->xc-2 << "\t" << prtclpass->yc-2 <<
"\t"
     << prtclpass->perimreal << "\t" << prtclpass->area << endl;
   prtclpass=prtclpass->nextprtcl;
 }
 return;
}

CSimupart::~CSimupart()
{
 particle *prtclpass, *prtcltemp;
 prtclpass=prtclhead;
 struct point *pointpass, *pointtemp, *pfillpass, *pfilltemp;
 while (prtclpass != NULL)
 {
   pointpass = prtclpass->pointhead;
   while (pointpass != NULL)
   {
     pointtemp = pointpass->nextpoint;
     delete pointpass;
     pointpass = pointtemp;
   }
   prtclpass->pointhead =pointpass;

   pfillpass = prtclpass->pfillhead;
   while (pfillpass != NULL)
   {
     pfilltemp = pfillpass->nextpoint;
     delete pfillpass;
     pfillpass = pfilltemp;
   }

   prtclpass->pfillhead =pfillpass;


   prtcltemp=prtclpass;
   prtclpass=prtcltemp->nextprtcl;
   delete prtcltemp;
 }
 return;
}


struct point {
 int x,y;
 struct point *nextpoint;
};

class particle {
```

```
public:
 struct point *pointhead;
 struct point *pfillhead;

 int area;
 int perim;
 double perimreal;
 double xc;
 double yc;
 particle *nextprtcl;
};

class CSimupart {
public:
 particle *prtclhead;
 CSimupart(void);
 CSimupart(char *particlefilename);
 ~CSimupart(void);
};

int xc_yc_area(particle *prtclhead,int perimax)
{//particle area info
 struct point1
 {
   int xx;
   int yy;
 };

 struct point1 *pnt1= new point1[int(perimax*perimax/4.0/Pi)];
 struct point1 *pnt2= new point1[int(perimax*perimax/4.0/Pi)];

 if (  pnt1 == NULL || pnt2 == NULL)
   return 2; //runs out of ram

 particle *prtclpass= prtclhead;

 int i,j,ip,count,yp,yp2,xp,jp,p1,p2,xp2,min,max,size;
 int *array=new int[perimax],*x2 = new int[perimax],*y2 = new
int[perimax],*x = new int[perimax],*y = new int[perimax];
 double up,tot_len,xc,yc,length;
 struct point *pointpass, *pfillpass, *pfilltemp;

 while (prtclpass !=NULL)
 {
   tot_len=0;
   up=0;
   size=prtclpass->perim;

   pointpass = prtclpass->pointhead;
   for (i=1;i<=size;i++)
   {
     x2[i]=x[i]=pointpass->x;
     y2[i]=y[i]=pointpass->y;
     pointpass = pointpass->nextpoint;
   }

   p1=p2=1;
```

```
    for (ip=1;ip<=size;ip++)
    {
      //calculate xc and area for an particle.
      for (i=1;i<perimax;i++)
        array[i]=0;
      count=1;
      yp=y[ip];
      array[1]=x[ip];
        if (yp==0)  continue;

      for (jp=ip+1;jp<=size;jp++)
      {
        yp2=y[jp];
        if (yp2==0) continue;

        if (yp2==yp)
        {
          count++;
       array[count]=x[jp];//putting X co-ordinate in an array.. for same
Ys
          y[jp]=0;
        }
      }

      min=array[1];
      max=array[1];
      for(i=1;i<=count;i++)
      {
        if (array[i]<min) min=array[i];
        if (array[i]>max) max=array[i];
      }
      xp=min;
      for (i=1;i<=(max-min+1);i++)
      {
        pnt1[p1].xx=xp; pnt1[p1].yy=yp;
        xp++;p1++;
      }
      length = max-min+1;
      xc=min+length/2;

      tot_len=tot_len+length;
      up=up+xc*length;
    }
    if (tot_len==0) tot_len=1;
    prtclpass->xc=up/tot_len;
    tot_len=0;
    up=0;
    for (i=1;i<=size;i++)
    {
      x[i]=x2[i];
      y[i]=y2[i];
    }

    for (ip=1;ip<=size;ip++)
    { //calculate yc and area for an particle.
      for (i=1;i<perimax;i++)
```

```
        array[i]=0;
        count=1;
        xp=x[ip];
        array[count]=y[ip];
        if (xp==0)
        continue;

      for (jp=ip+1;jp<=size;jp++)
      {
        xp2=x[jp];
        if (xp2==0)
          continue;

        if (xp2==xp)
        {
          count=count+1;
          array[count]=y[jp];//putting Y co-ordinate in an array.. for
same Xs
          x[jp]=0;
        }
      }

      min=array[1];
      max=array[1];

      for(i=1;i<=count;i++)
      {
        if (array[i]<min) min=array[i];
        if (array[i]>max) max=array[i];
      }

      yp=min;

      for (i=1;i<=(max-min+1);i++)
      {
        pnt2[p2].xx=xp;pnt2[p2].yy=yp;   // filling of the particle
(line by line)
        yp++;p2++;
      }
      length = max-min+1;
      yc=min+length/2;
      tot_len=tot_len+length;
      up=up+yc*length;
    }
    if (tot_len==0) tot_len=1;
    prtclpass->yc=up/tot_len;

    pfillpass = prtclpass->pfillhead = NULL;

    for (i=1;i<=p1-1;i++)
    for (j=1;j<=p2-1;j++)
    {
      if (pnt1[i].xx==pnt2[j].xx  && pnt1[i].yy==pnt2[j].yy)
      {
        prtclpass->area++;
        pfilltemp = new point;
        pfilltemp->x = pnt1[i].xx;
```

```
        pfilltemp->y = pnt1[i].yy;
        pfilltemp->nextpoint = NULL;

        if(prtclpass->pfillhead == NULL)
          prtclpass->pfillhead = pfilltemp;
        else
          pfillpass->nextpoint=pfilltemp;
        pfillpass=pfilltemp;
        continue;
      }
    }
    prtclpass=prtclpass->nextprtcl;
 }//end while

 delete [] pnt1;
 delete [] pnt2;
 delete [] x;
 delete [] x2;
 delete [] y;
 delete [] y2;
 delete [] array;
 return 1;
}
```

B.6 3D Connected Component Labeling

```
/*********************************************************************
 The code is modified from 2D connected component labeling code
originally created by Tola, Engin. 2006 June 12.
  Homepage. <http://cvlab.epfl.ch/~tola/index.htm>"
*********************************************************************/

void CImageMaxApp::OnGetParticleShape()
{//get 3d particle shapes
 dlg3DSimu dlg;
 if (dlg.DoModal()==IDOK)
 {
   char sBuffer[BUFFERLEN];

   CFileDialog
dlgFile(TRUE,NULL,NULL,OFN_ALLOWMULTISELECT|OFN_EXPLORER|OFN_ENABLESIZI
NG,"Data Files (*.bmp;*.tif;*.png)|*.bmp; *.tif; *.png|All Files
(*.*)|*.*||");
   dlgFile.m_ofn.lpstrFile = sBuffer;
   dlgFile.m_ofn.lpstrFile[0] = '\0';
   dlgFile.m_ofn.nMaxFile = BUFFERLEN;
   //get mru, if not exist, use "my documents"
   CString m_sOpenFileDir= GetProfileString("Settings", "OpenFileDir",
"%USERPROFILE%\\My Documents");
   dlgFile.m_ofn.lpstrInitialDir=(LPCSTR) m_sOpenFileDir;

   if (dlgFile.DoModal() == IDOK)
   {
```

```
      int filenum=0;
      CImageMaxDoc *NewDoc=(CImageMaxDoc*)((CImageMaxApp*)AfxGetApp())-
>demoTemplate->OpenDocumentFile(NULL);
      //NewDoc->image = new CxImage();

      POSITION pos = dlgFile.GetStartPosition();
      while (pos)
      {
        NewDoc->m_Filename_array[filenum] =
dlgFile.GetNextPathName( pos );
        filenum++;
      }

      //save mru
      m_sOpenFileDir= NewDoc->m_Filename_array[0];

      int slashpos = m_sOpenFileDir.ReverseFind('\\');
      if(slashpos != -1)
      {
        m_sOpenFileDir = m_sOpenFileDir.Left(slashpos);
      }
      WriteProfileString("Settings", "OpenFileDir", m_sOpenFileDir );

      SetCurrentDirectory(m_sOpenFileDir);

      //order filename
      BubbleSort(NewDoc->m_Filename_array, filenum);

      if (!NewDoc->OnOpenDocument(NewDoc->m_Filename_array[0]))
        return;
      NewDoc->m_fp[4]=(void *)(int)dlg.m_RemoveOnBoarder;
      NewDoc->m_fp[3]=(void *)(int)dlg.m_2pointR;
      NewDoc->m_fp[2]=(void *)(int)dlg.m_E2point;
          NewDoc->m_fp[1]=(void *)(int)dlg.m_LabelNumber;
      NewDoc->m_fp[0]=(void *)(int)filenum;
      NewDoc->m_MenuCommand=ID_GetParticleShape;
      NewDoc->m_fPixelSize=dlg.m_PixelSize;
      NewDoc->m_fZDis=dlg.m_ZDis;


      NewDoc->hThread=(HANDLE)_beginthread(RunSimuThread,0,NewDoc);

      CString s;
      s.Format("3D");
      NewDoc->SetTitle(s);
      NewDoc->UpdateAllViews(0,WM_USER_NEWIMAGE);
    }
  }
}

void CImageMaxApp::OnSizedistribute()
{
    CFileDialog
dlgFile1(TRUE,"txt","size_distribute.txt",OFN_FILEMUSTEXIST,"Data Files
(*.txt;*.dat)|*.txt; *.dat|All Files (*.*)|*.*||");
```

```
 CFileDialog
dlgFile(TRUE,"txt","results_voxels.txt.gz",OFN_FILEMUSTEXIST,"Data
Files (*.txt;*.dat;*.gz)|*.txt; *.dat; *.gz|All Files (*.*)|*.*||");

 dlgFile.m_ofn.lpstrTitle="open the particle voxel data file";
 dlgFile1.m_ofn.lpstrTitle="open the size distribution data file";

 CString m_sOpenFileDir= GetProfileString("Settings", "OpenFileDir",
"%USERPROFILE%\\My Documents");


 if (dlgFile1.DoModal() == IDOK)
 if (dlgFile.DoModal() == IDOK)
 {
   CImageMaxDoc *NewDoc=(CImageMaxDoc*)((CImageMaxApp*)AfxGetApp())-
>demoTemplate->OpenDocumentFile(NULL);
   NewDoc->m_fFilename = dlgFile.GetPathName();
   NewDoc->m_fFilename2 = dlgFile1.GetPathName();

   CxImage *newout = new CxImage(150,50,24);
   NewDoc->image = newout;
   NewDoc->image->SetStdPalette();

   m_sOpenFileDir= NewDoc->m_fFilename;
   int slashpos = m_sOpenFileDir.ReverseFind('\\');
   if(slashpos != -1)
   {
     m_sOpenFileDir = m_sOpenFileDir.Left(slashpos);
   }
   WriteProfileString("Settings", "OpenFileDir", m_sOpenFileDir );
   SetCurrentDirectory(m_sOpenFileDir);

   NewDoc->m_MenuCommand=ID_SIZEDISTRIBUTE;
   NewDoc->hThread=(HANDLE)_beginthread(RunSimuThread,0,NewDoc);

   CString s;
   s.Format("Size Distribute");
   NewDoc->SetTitle(s);
   NewDoc->UpdateAllViews(0,WM_USER_NEWIMAGE);
 }

}

void RunSimuThread(void *lpParam)
{
 POSITION posView;
 CView *pView;

 CImageMaxDoc *pDoc = (CImageMaxDoc *)lpParam;

 if (pDoc==NULL) return;
 if (pDoc->image==NULL) return;

  //prepare for elaboration
 pDoc->image->SetProgress(0);
 pDoc->image->SetEscape(0);
```

```
 nSimu2d=0;
 pDoc->Stopwatch(0);

 pDoc->hProgress = (HANDLE)_beginthread(RunSimuProgressThread,0,pDoc);

 switch (pDoc->m_MenuCommand)
 {
     case ID_GetParticleShape://get particle shape
     {
       int i,j;
       pDoc->MouseMoveUpdate=false; //disable onmousemove
       int depth = (int)pDoc->m_fp[0];
       int LabelNumber = (int)pDoc->m_fp[1];
       int m = (int)pDoc->m_fp[3];
       int E2point = (int)pDoc->m_fp[2];
       bool removeOnBoarder=((int)pDoc->m_fp[4] == 0 )? false:true;

       DWORD width=pDoc->image->GetWidth();
       DWORD height=pDoc->image->GetHeight();

       double zDis = pDoc->m_fZDis/pDoc->m_fPixelSize;

       BYTE * im3D=new BYTE[width*height*depth] ;
       BYTE *im = new BYTE[width*height] ;
       nSimupart = 0;

       //copy 2d sections into 3D array
       for (i=0; i<depth;i++)
       {
         posView = pDoc->GetFirstViewPosition();
         pView = pDoc->GetNextView(posView);

         if (!pDoc->LoadImageThread(pDoc->m_Filename_array[i]))
           break;

         pDoc->image->DecreaseBpp(8,false);
         pDoc->image->GetGrayHead(im);
         memcpy(&im3D[i*height*width],im,height*width);
         SendMessage(pView->m_hWnd, WM_USER_NEWDRAWS,0,0);
         nSimupart++;
       }
       delete [] im;
       CreateDirectory("segmented", NULL);
       K3DConnectedComponentLabeler* ccl = new
K3DConnectedComponentLabeler();

       ccl->InitConfig(1);
       ccl->removeonboarder=removeOnBoarder;
       // mask to be processed
       if (ccl->SetMask(im3D, width, height, depth)==1)
       {//not enough ram
         delete [] im3D;
         delete ccl;
         break;
       }

       delete [] im3D;
```

```
        int n = ccl->Process(); // Main function

        if (n == 2)
        {
          AfxMessageBox( "Too many particles!");
          delete ccl;
          break;
        }

        ccl->unSetMask();

        vector<K3DConnectedComponentLabeler::KBox> objects = ccl-
>m_Components;
        int objnumber = int( objects.size()) ;

        //save image
        int imSize = width*height*depth;
        DWORD* imout = new DWORD[imSize];// output mask
        for(i=0; i<imSize; i++)
          imout[i] = 0;

        FILE *fpout;
        ofstream ofstream_("segmented\\results_voxels.txt.gz",ios::out |
ios::binary );
        zlib_stream::zip_ostream fpoutvoxel(ofstream_,true);

        fpout = fopen("segmented\\results_size.txt","w");
        fprintf(fpout,"total number: %d",objnumber);
        fprintf(fpout,"\nnumber\tsize\tR\tG\tB");
        fpoutvoxel << objnumber;
        RGBQUAD particleColor;
        for (i = 0; i<objnumber; i++)
        {
          particleColor= pDoc->image->RGBtoRGBQUAD(objects[i].ID*2222);

fprintf(fpout,"\nNo.%d\t%f\t%d\t%d\t%d",i,objects[i].Area*pDoc-
>m_fZDis*pDoc->m_fPixelSize*pDoc->m_fPixelSize,

particleColor.rgbRed,particleColor.rgbGreen,particleColor.rgbBlue);

          vector<K3DConnectedComponentLabeler::KVoxel> Voxel =
objects[i].m_Voxel;
          vector<K3DConnectedComponentLabeler::KVoxel>::iterator iter;
          for (iter=Voxel.begin(); iter!=Voxel.end(); iter++)
          {
            int index = iter->z*width*height + iter->y*width +iter->x;
            imout[index]=objects[i].ID;
            fpoutvoxel << "\n" << iter->x+objects[i].centerx() << "\t"
<< iter->y+objects[i].centery() << "\t"
<< iter->z+objects[i].centerz();
          }
          fpoutvoxel << "\n" << "-100000";

        }

        fclose(fpout);
        fpoutvoxel.zflush();
```

```
        pDoc->image->IncreaseBpp(24);
        for (i=0; i<depth;i++)
        {
          CString zeros, imgNString, imagename;
          if (i<10)
            zeros = "00";
          if ((i<100) && (i>=10))
            zeros = "0";
          if (i>=100)
            zeros = "";

          imgNString.Format("%d", i);
          imagename ="segmented\\Result2_" + zeros + imgNString + ".png";
          pDoc->image->SetColorHead(&imout[i*height*width]);

          if (LabelNumber)
          {
            for (j = 0; j<objnumber; j++)
            {
              if (objects[j].top>=i && objects[j].bottom<=i)
              {
                int Labelx =
(objects[j].topLeft.x+objects[j].bottomRight.x)/2;
                int Labely =
(objects[j].topLeft.y+objects[j].bottomRight.y)/2;
                CString s;
                s.Format("%d",objects[j].ID);

                pDoc->image->DrawString(0,Labelx,Labely,s,
                            pDoc->image->RGBtoRGBQUAD(RGB(255,255,255)),
                            "New Arial", 16, 300);
              }
            }
          }
          pDoc->image->Save(imagename, CXIMAGE_FORMAT_PNG);
          nSimupart++;
        }
        delete ccl;
        delete [] imout;

        pDoc->MouseMoveUpdate=true; //enable onmousemove
        break;
      }//end case ID_GetParticleShape
    case ID_SIZEDISTRIBUTE:
      {
        nSimu2d=0;
        char* filename =  (char*)(const char*)pDoc->m_fFilename;
        char* filename2 =  (char*)(const char*)pDoc->m_fFilename2;
        FILE *fp;
        int i,j;
        if( (fp  = fopen(filename2, "r" )) == NULL )
        {
          AfxMessageBox("Cant open size distribute data!");
          break;
        }
        char sdummy[20];
```

```
int binnumber,objnumberNew;
fscanf(fp,"%s %d  %d",sdummy,&binnumber,&objnumberNew);

int *bin = new int[binnumber];
int *binsize = new int[binnumber];
int *binvalue= new int[binnumber];

for (i=0;i<binnumber;i++)
{
  fscanf(fp,"%d %d",&bin[i],&binsize[i]);
  binvalue[i]=0;
}

fclose(fp);

//get particle voxel
vector<K3DConnectedComponentLabeler::KBox> objects;

ifstream ifstream_;
ifstream_.open(filename, ios::in | ios::binary);
if (!ifstream_.is_open())
{
  AfxMessageBox("Cant open voxel data!");
  break;
}

// create unzipper istream
zlib_stream::zip_istream fpzipper( ifstream_);

int objnumber,x,y,z,tmp;//,averageArea;p_width,p_height,
fpzipper>>objnumber;

for(i=0; i<objnumber; i++)
{
  K3DConnectedComponentLabeler::KBox newComponent;

  newComponent.ID = i+1;
  newComponent.bottomRight = CPoint(INT_MIN, INT_MAX);
  newComponent.topLeft     = CPoint(INT_MAX, INT_MIN);
  newComponent.top = INT_MIN;
  newComponent.bottom = INT_MAX;
  newComponent.Area = 0;

  while (1)
  {
    //fscanf(fp,"%d",&tmp);
    fpzipper>>tmp;
    if (tmp==-100000)
      break;

    //z = tmp/p_width/p_height;
    //int zremain = tmp%(p_width*p_height);
    //x = zremain%p_width;
    //y = zremain/p_width;
    x = tmp;
    fpzipper>>y>>z;
```

```
            if( z < newComponent.bottom )
              newComponent.bottom = z;

            if( z > newComponent.top )
              newComponent.top = z;

            if( x > newComponent.bottomRight.x )
              newComponent.bottomRight.x = x;

            if( x < newComponent.topLeft.x     )
              newComponent.topLeft.x     = x;

            if( y < newComponent.bottomRight.y )
              newComponent.bottomRight.y = y;

            if( y > newComponent.topLeft.y     )
              newComponent.topLeft.y     = y;

            K3DConnectedComponentLabeler::KVoxel newVoxel;
            newVoxel.x = x;
            newVoxel.y = y;
            newVoxel.z = z;
            newComponent.m_Voxel.push_back(newVoxel);
            newComponent.Area++;
          }
          for (j=0;j<binnumber;j++)
          {
            if (newComponent.Area<bin[j] && newComponent.Area>bin[j-1])
            {
              if(binvalue[j]<binsize[j])
              {
                objects.push_back(newComponent);
                binvalue[j]++;
                break;
              }
            }
          }
        }
        for (j=0;j<binnumber;j++)
          if (binvalue[j]==0)
            objnumberNew-=binsize[j];

        FILE *fpout;//,*fpoutvoxel;
        ofstream ofstream_("results_voxels_new.txt.gz",ios::out |
ios::binary );
        zlib_stream::zip_ostream fpoutvoxel(ofstream_,true);

        fpout = fopen("results_size_new.txt","w");
        fprintf(fpout,"total number: %d",objnumberNew);
        fpoutvoxel << objnumberNew;

        for (i = 0; i<objects.size(); i++)
        {
          fprintf(fpout,"\nNo.%d\t%d",i,objects[i].Area);
```

```
          vector<K3DConnectedComponentLabeler::KVoxel> Voxel =
objects[i].m_Voxel;
          vector<K3DConnectedComponentLabeler::KVoxel>::iterator iter;
          for (iter=Voxel.begin(); iter!=Voxel.end(); iter++)
          {
              fpoutvoxel << "\n" << iter->x << "\t" << iter->y << "\t"
<< iter->z;
          }
          fpoutvoxel << "\n" << "-100000";

        }


        for (j=0;j<binnumber;j++)
        {
          while(binvalue[j]<binsize[j] && binvalue[j]!=0)
          {
              for (i = 0; (i<objects.size()&&binvalue[j]<binsize[j] );
i++)
            {
              if (objects[i].Area<bin[j] && objects[i].Area>bin[j-1])
              {
                fprintf(fpout,"\nNo.%d\t%d",i,objects[i].Area);
                vector<K3DConnectedComponentLabeler::KVoxel> Voxel =
objects[i].m_Voxel;
                vector<K3DConnectedComponentLabeler::KVoxel>::iterator
iter;
                for (iter=Voxel.begin(); iter!=Voxel.end(); iter++)
                {
                  //int index = iter->z*p_width*p_height + iter-
>y*p_width +iter->x;
                  //fprintf(fpoutvoxel,"\n%d",index);
                  //fpoutvoxel << "\n" << index;
                  fpoutvoxel << "\n" << iter->x << "\t" << iter->y <<
"\t" << iter->z;
                }
                //fprintf(fpoutvoxel,"\n-100");
                fpoutvoxel << "\n" << "-100000";
                binvalue[j]++;

              }
            }
          }
        }

        fclose(fpout);
        fpoutvoxel.zflush();
        //fclose(fpoutvoxel);
        delete []bin;
        delete []binvalue;
        delete []binsize;

        objects.clear();

        pDoc->image->Clear();
        pDoc->image->DrawString(0,5,20,"Done!",
              pDoc->image->RGBtoRGBQUAD(RGB(0,255,0)),
```

```
            "New Arial", 16, 300);
       posView = pDoc->GetFirstViewPosition();
       pView = pDoc->GetNextView(posView);
       SendMessage(pView->m_hWnd, WM_USER_NEWDRAWS,0,0);
       break;
     }//end case ID_SIZEDISTRIBUTE
   }

 //pDoc->image->SetProgress(100);
 nSimu2d=100;
 pDoc->Stopwatch(1);

 pDoc->hThread=0;
 _endthread();
 return ;
}

class K3DConnectedComponentLabeler
{
private:
 class KNode
 {
 public:
   KNode();
   virtual ~KNode();
   KNode*  ngNext;
   KNode*  sgNext;
   int data;
 };

public:
 class KVoxel
 {
 public:
   short x,y,z;
   int ID;
 };

 class KBox
 {
 public:
   KBox();
   virtual ~KBox();
   CPoint topLeft;
   CPoint bottomRight;
   int top,bottom;
   int    ID;
   int    Area;
   int    centerx(int cx=0) { return cx-(topLeft.x+bottomRight.x)/2; }
   int    centery(int cy=0) { return cy-(topLeft.y+bottomRight.y)/2; }
   int    centerz(int cz=0) { return cz-(top+bottom)/2; }
   int    fill3D(DWORD *im, int cx, int cy, int cz, int width, int
height, int depth, bool fill=false, bool clear=false);
   int   fill3DRotate(DWORD *im, int cx, int cy, int cz, int width, int
height, int depth,double phi, double theta, double psi, int overlapped,
bool fill=false);
```

```
    vector<KVoxel> m_Voxel;
    BYTE *m_ArrayVoxel; // for rotation;
    int m_Boxwidth()    { return bottomRight.x-topLeft.x+1; }
    int m_Boxheight() { return topLeft.y-bottomRight.y+1; }
    int m_Boxdepth()  { return top-bottom+1;}
  };

private:
  class KLinkedList
  {
  public:
    void printTable();

    KNode * header;
    int  regionCount;

    void Search(int data, KNode* &p);

    void InsertData(int data);
    void InsertData(int addGroup,int searchGroup);

    KLinkedList();
    ~KLinkedList();
  };
  int GetLabel(int i, int j, int k=0);
public:
  DWORD*  GetOutput();

  int   m_ObjectNumber;
  DWORD*  m_MaskArray;

  int   m_nAreaThreshold;
  int   m_height;
  int   m_width;
  int     m_depth;
  bool    removeonboarder;

  vector<KBox> m_Components;
  void  Clear();
  void  InitConfig( int AreaThreshold );
  int Process();
  int SetMask(BYTE* mask, int width, int height, int depth);
  void  unSetMask();
  K3DConnectedComponentLabeler();
  virtual ~K3DConnectedComponentLabeler();
};

#define HARDLIMITNUMBER 4294967295

////////////////////////////////////////////////////////////////////////
// Construction/Destruction
////////////////////////////////////////////////////////////////////////

K3DConnectedComponentLabeler::K3DConnectedComponentLabeler()
{
  m_MaskArray = NULL;
  m_height = 0;
```

```
 m_width  = 0;
 m_nAreaThreshold = 0;
 m_ObjectNumber = 0;
 removeonboarder=true;
}

K3DConnectedComponentLabeler::~K3DConnectedComponentLabeler()
{
 m_Components.clear();
 unSetMask();
}

void K3DConnectedComponentLabeler::InitConfig(int nAreaThreshold)
{
 m_nAreaThreshold = nAreaThreshold;
}

int K3DConnectedComponentLabeler::GetLabel(int i, int j, int k)
{
 if(m_MaskArray == NULL) return 0;

 if (i<0 || i>=m_height || j<0 || j>=m_width || k<0 || k >=m_depth)
return 0;

 return m_MaskArray[k*m_height*m_width+i*m_width+j];

}
int K3DConnectedComponentLabeler::Process()
{
 Clear();
 if(m_MaskArray == NULL) return 1;

// Binarize(); // if pixel_intensity > 0 --> pixel_intensity = 1;

 KLinkedList eqTable;

 KNode * tmp = NULL;
 KNode * p = NULL;
 KNode * p2  = NULL;

 int i,k;
 int label = 2;
 int index = 1;
 int north, west, nWest, nEast, top[9];


 int    * regionLabel = NULL;
 int    * lookUpTable = NULL;
 long int * regionArea  = NULL;

 int regionNumber;


 int sliceSize = m_height*m_width;

 int data=0;
 for(k=0; k<m_depth; k++) {
```

```
nSimu2d=float(k)/m_depth*50;
for(i=0; i<m_height; i++) {
  for(int j=0; j<m_width; j++)
  {
    index = k*sliceSize+i*m_width+j;

    if(m_MaskArray[index] == 1)
    {
      top[0] = GetLabel(i-1,j-1,k-1);  // 6 7 8
      top[1] = GetLabel(i-1,j,k-1);    // 3 4 5
      top[2] = GetLabel(i-1,j+1,k-1);  // 0 1 2
      top[3] = GetLabel(i,j-1,k-1);
      top[4] = GetLabel(i,j,k-1);
      top[5] = GetLabel(i,j+1,k-1);
      top[6] = GetLabel(i+1,j-1,k-1);
      top[7] = GetLabel(i+1,j,k-1);
      top[8] = GetLabel(i+1,j+1,k-1);

      west = GetLabel(i,j-1,k);
      north = GetLabel(i-1,j,k);
      nEast = GetLabel(i-1,j+1,k);
      nWest = GetLabel(i-1,j-1,k);


      // after finding the neighbour labels
      if ( west > 1 )
      {
        m_MaskArray[index] = west;

        if( nWest>1 )
          eqTable.InsertData(west,nWest);

        if( north>1 && nWest<=1 )
          eqTable.InsertData(west,north);

        if( nEast>1 && north<=1 )
          eqTable.InsertData(west,nEast);

        if (top[4]>1) eqTable.InsertData(west,top[4]);

      }
      else if( nWest > 1)
      {
        m_MaskArray[index] = nWest;

        if ( north<=1 && nEast>1 )
          eqTable.InsertData(nWest,nEast);

        if (top[4]>1) eqTable.InsertData(nWest,top[4]);
      }
      else if( north > 1 )
      {
        m_MaskArray[index] = north;
        if (top[4]>1) eqTable.InsertData(north,top[4]);
      }
      else if( nEast > 1 )
      {
```

```
            m_MaskArray[index] = nEast;
            if (top[4]>1) eqTable.InsertData(nEast,top[4]);
          }
          else if (top[4]>1)
          {
              m_MaskArray[index] = top[4];
          }
          else
          {
            m_MaskArray[index] = label;
            eqTable.InsertData(label);
            label++;

            if (label>HARDLIMITNUMBER) return 2;
          }
          //link to the previous slice
          if (top[4]<=1) {
          if (top[5]>1) {
            eqTable.InsertData(m_MaskArray[index],top[5]);
            if (top[3]>1) {
                eqTable.InsertData(m_MaskArray[index],top[3]);
            } else {
              if (top[6]>1)
{ eqTable.InsertData(m_MaskArray[index],top[6]);}
              if (top[0]>1)
{ eqTable.InsertData(m_MaskArray[index],top[0]);}
            }
          } else if (top[3]>1) {
            eqTable.InsertData(m_MaskArray[index],top[3]);
            if (top[8]>1)
{ eqTable.InsertData(m_MaskArray[index],top[8]);}
            if (top[2]>1)
{ eqTable.InsertData(m_MaskArray[index],top[2]);}
          } else if (top[1]>1) {
            eqTable.InsertData(m_MaskArray[index],top[1]);
            if (top[7]>1) {
                eqTable.InsertData(m_MaskArray[index],top[7]);
            } else {
              if (top[6]>1)
{ eqTable.InsertData(m_MaskArray[index],top[6]);}
              if (top[8]>1)
{ eqTable.InsertData(m_MaskArray[index],top[8]);}
            }

          } else if (top[0]>1) {
            if (top[2]>1)
{ eqTable.InsertData(m_MaskArray[index],top[2]);}
            eqTable.InsertData(m_MaskArray[index],top[0]);
            if (top[7]>1) {
                eqTable.InsertData(m_MaskArray[index],top[7]);
            } else {
              if (top[6]>1)
{ eqTable.InsertData(m_MaskArray[index],top[6]);}
              if (top[8]>1)
{ eqTable.InsertData(m_MaskArray[index],top[8]);}
            }
```

```
          } else if (top[2]>1) {
            eqTable.InsertData(m_MaskArray[index],top[2]);
            if (top[7]>1) {
                eqTable.InsertData(m_MaskArray[index],top[7]);
            } else {
              if (top[6]>1)
{ eqTable.InsertData(m_MaskArray[index],top[6]);}
              if (top[8]>1)
{ eqTable.InsertData(m_MaskArray[index],top[8]);}
            }
          } else if (top[7]>1) {
            eqTable.InsertData(m_MaskArray[index],top[7]);
          } else if (top[6]>1) {
            eqTable.InsertData(m_MaskArray[index],top[6]);
            if (top[8]>1)
{ eqTable.InsertData(m_MaskArray[index],top[8]);}
          } else if (top[8]>1) {
            eqTable.InsertData(m_MaskArray[index],top[8]);
          }
          }
        }
      }
    }
  }
 regionNumber = eqTable.regionCount;

 if( regionNumber > 0 )
 {
   regionLabel  = new int[regionNumber];
   regionArea   = new long int[label];

   for(i=0; i<label; i++ )
     regionArea[i]=0;

   tmp = eqTable.header;
   i=0;
   do
   {
     regionLabel[i]=tmp->data;
     tmp=tmp->ngNext;
     i=i+1;
   }
   while(tmp!=NULL);

   lookUpTable = new int[label];

   p  = eqTable.header;
   p2 = p;

   do
   {
     data=p->data;
     do
     {
       lookUpTable[p2->data] = data;
       p2 = p2->sgNext;
     }
```

```
  while(p2 != NULL);
  p  = p->ngNext;
  p2 = p;
}
while(p != NULL);

for(k=0; k<m_depth; k++) {
  for (i=0;i<m_height; i++  ) {
    for (int j=0; j<m_width; j++ )
    {
      index=k*sliceSize+i*m_width+j;

      if( m_MaskArray[index]>1 )
      {
        data=lookUpTable[ m_MaskArray[index] ];
        m_MaskArray[index]=data;

        regionArea[ data ]++;
      }
      else
        m_MaskArray[index]=0;
    }
  }
}
int* trueLabelArray = new int[label];
for(i=0; i<label; i++)
{
  trueLabelArray[i] = 0;

  KBox newComponent;
  newComponent.ID = i+1;
  newComponent.bottomRight = CPoint(INT_MIN, INT_MAX);
  newComponent.topLeft     = CPoint(INT_MAX, INT_MIN);
  m_Components.push_back(newComponent);
}

m_ObjectNumber = 0;

int nImSize = sliceSize*m_depth;
int x, y, z;
KVoxel newVoxel;

for(i=0; i<nImSize; i++)
{
  nSimu2d=float(i)/nImSize*50+49;
  z = i/m_width/m_height;
  int zremain = i%(m_width*m_height);
  x = zremain%m_width;
  y = zremain/m_width;


  if( regionArea[ m_MaskArray[i] ] <= m_nAreaThreshold )
    m_MaskArray[i] = 0;
  else
  {
    if( trueLabelArray[ m_MaskArray[i] ] == 0 )
    {
```

```
          m_ObjectNumber++;

          m_Components[ m_ObjectNumber-1 ].topLeft.x     = x;
          m_Components[ m_ObjectNumber-1 ].topLeft.y     = y;
          m_Components[ m_ObjectNumber-1 ].bottomRight.x = x;
          m_Components[ m_ObjectNumber-1 ].bottomRight.y = y;
          m_Components[ m_ObjectNumber-1 ].top = z;
          m_Components[ m_ObjectNumber-1 ].bottom = z;
          m_Components[ m_ObjectNumber-1 ].Area =
regionArea[ m_MaskArray[i] ] ;

          trueLabelArray[ m_MaskArray[i] ] = m_ObjectNumber;
          m_MaskArray[i] = m_ObjectNumber;

          newVoxel.x = x;
          newVoxel.y = y;
          newVoxel.z = z;

          m_Components[ m_ObjectNumber-1 ].m_Voxel.push_back(newVoxel);

        }
        else
        {
          m_MaskArray[i] = trueLabelArray[ m_MaskArray[i] ];

          if( z < m_Components[ m_MaskArray[i]-1 ].bottom )
            m_Components[ m_MaskArray[i]-1 ].bottom = z;

          if( z > m_Components[ m_MaskArray[i]-1 ].top )
            m_Components[ m_MaskArray[i]-1 ].top = z;

          if( x > m_Components[ m_MaskArray[i]-1 ].bottomRight.x )
            m_Components[ m_MaskArray[i]-1 ].bottomRight.x = x;

          if( x < m_Components[ m_MaskArray[i]-1 ].topLeft.x     )
            m_Components[ m_MaskArray[i]-1 ].topLeft.x     = x;

          if( y < m_Components[ m_MaskArray[i]-1 ].bottomRight.y )
            m_Components[ m_MaskArray[i]-1 ].bottomRight.y = y;

          if( y > m_Components[ m_MaskArray[i]-1 ].topLeft.y     )
            m_Components[ m_MaskArray[i]-1 ].topLeft.y     = y;

          newVoxel.x = x;
          newVoxel.y = y;
          newVoxel.z = z;
          m_Components[ m_MaskArray[i]-1 ].m_Voxel.push_back(newVoxel);
        }
      }
    }

    while( m_Components.size() != m_ObjectNumber )
      m_Components.pop_back();


    delete trueLabelArray; trueLabelArray = NULL;
```

```
   vector<KBox> tmp_Components;
   if (removeonboarder)
   {
     while (!m_Components.empty())
     {
       if (m_Components.back().top==m_depth-1 ||
m_Components.back().bottom==0 || m_Components.back().topLeft.x ==0 ||
m_Components.back().topLeft.y == m_height-1 ||
         m_Components.back().bottomRight.x == m_width-1 ||
m_Components.back().bottomRight.y == 0)
         m_Components.pop_back();
       else
       {
         tmp_Components.push_back(m_Components.back());
         m_Components.pop_back();
       }
     }
     m_Components=tmp_Components;
   }
}

 delete []lookUpTable;
 lookUpTable = NULL;

 delete []regionArea;
 regionArea = NULL;

 delete []regionLabel;
 regionLabel = NULL;

 return 0;
}


//////// private KNode implementations ////////////////////

K3DConnectedComponentLabeler::KNode::KNode()
{
 this->data=0;
 this->sgNext=NULL;
 this->ngNext=NULL;
}

K3DConnectedComponentLabeler::KNode::~KNode()
{
}

K3DConnectedComponentLabeler::KBox::KBox()
{
 ID = 0;
 bottomRight = 0;
 topLeft = 0;
 top = 0;
 bottom = 0;
 Area = 0;
}
```

```
K3DConnectedComponentLabeler::KBox::~KBox()
{
 m_Voxel.clear();
}

int K3DConnectedComponentLabeler::KBox::fill3D(DWORD *im, int cx, int
cy, int cz, int width, int height, int depth, bool fill, bool clear)
{
 int filled=0, overlap=0;
 int x,y,z;

 vector<KVoxel>::iterator iter;
 for (iter=m_Voxel.begin(); iter!=m_Voxel.end(); iter++)
 {
   x=iter->x+centerx(cx);
   y=iter->y+centery(cy);
   z=iter->z+centerz(cz);

   //if ( (x<0) || (x>=width) || (y<0) || (y>=height) || (z<0) ||
(z>=depth)) return 1000000; //no particles on box boarder.

   if (x<0) x+=width;
   if (x>=width) x-=width;

   if (y<0) y+=height;
   if (y>=height) y-=height;

   if (z<0) z+=depth;
   if (z>=depth) z-=depth;


   int index = z*width*height + y*width +x;

   if (clear)
   {
     im[index]-=ID;
     if (im[index]==0)
       filled++;
   }
   else
   {
     if (im[index]>0)
     {
       overlap++;
       if(fill)
         im[index]+=ID;
     }
     else
     {
       if(fill)
       {
         im[index]=ID;
         filled++;
       }
     }
   }
 }
```

```
 if (clear) return filled;
 if (fill)
   return filled;
 else
   return overlap;
}

int K3DConnectedComponentLabeler::KBox::fill3DRotate(DWORD *im, int cx,
int cy, int cz, int width, int height, int depth,
                          double phi, double theta, double psi, int
overlapped, bool fill)
{
 int filled=0, overlap=0;
 int x,y,z,xnew,ynew,znew;
 double xold,yold,zold;
 double c11,c12,c13,c21,c22,c23,c31,c32,c33;
 rand_rotation(phi,psi,theta,&c11,&c12,&c13,
               &c21,&c22,&c23,
               &c31,&c32,&c33);


 // Calculate the size of the new box
 KVoxel p1,p2,p3,p4,p5,p6,p7,p8;
 p1.x=topLeft.x;
 p1.y=bottomRight.y;
 p1.z=bottom;
 p2.x=topLeft.x;
 p2.y=topLeft.y;
 p2.z=bottom;
 p3.x=bottomRight.x;
 p3.y=topLeft.y;
 p3.z=bottom;
 p4.x=bottomRight.x;
 p4.y=bottomRight.y;
 p4.z=bottom;
 p5.x=topLeft.x;
 p5.y=bottomRight.y;
 p5.z=top;
 p6.x=topLeft.x;
 p6.y=topLeft.y;
 p6.z=top;
 p7.x=bottomRight.x;
 p7.y=topLeft.y;
 p7.z=top;
 p8.x=bottomRight.x;
 p8.y=bottomRight.y;
 p8.z=top;


 KVoxel newP1,newP2,newP3,newP4,newP5,newP6,newP7,newP8,
newUp,newBottom;

 newP1.x = int(c11*p1.x+c12*p1.y+c13*p1.z);
 newP1.y = int(c21*p1.x+c22*p1.y+c23*p1.z);
 newP1.z = int(c31*p1.x+c32*p1.y+c33*p1.z);

 newP2.x = int(c11*p2.x+c12*p2.y+c13*p2.z);
```

```
 newP2.y = int(c21*p2.x+c22*p2.y+c23*p2.z);
 newP2.z = int(c31*p2.x+c32*p2.y+c33*p2.z);

 newP3.x = int(c11*p3.x+c12*p3.y+c13*p3.z);
 newP3.y = int(c21*p3.x+c22*p3.y+c23*p3.z);
 newP3.z = int(c31*p3.x+c32*p3.y+c33*p3.z);

 newP4.x = int(c11*p4.x+c12*p4.y+c13*p4.z);
 newP4.y = int(c21*p4.x+c22*p4.y+c23*p4.z);
 newP4.z = int(c31*p4.x+c32*p4.y+c33*p4.z);

 newP5.x = int(c11*p5.x+c12*p5.y+c13*p5.z);
 newP5.y = int(c21*p5.x+c22*p5.y+c23*p5.z);
 newP5.z = int(c31*p5.x+c32*p5.y+c33*p5.z);

 newP6.x = int(c11*p6.x+c12*p6.y+c13*p6.z);
 newP6.y = int(c21*p6.x+c22*p6.y+c23*p6.z);
 newP6.z = int(c31*p6.x+c32*p6.y+c33*p6.z);

 newP7.x = int(c11*p7.x+c12*p7.y+c13*p7.z);
 newP7.y = int(c21*p7.x+c22*p7.y+c23*p7.z);
 newP7.z = int(c31*p7.x+c32*p7.y+c33*p7.z);

 newP8.x = int(c11*p8.x+c12*p8.y+c13*p8.z);
 newP8.y = int(c21*p8.x+c22*p8.y+c23*p8.z);
 newP8.z = int(c31*p8.x+c32*p8.y+c33*p8.z);

 newBottom.x =
min(min(min(newP1.x,newP2.x),min(newP3.x,newP4.x)),min(min(newP5.x,newP
6.x),min(newP7.x,newP8.x)));
 newBottom.y =
min(min(min(newP1.y,newP2.y),min(newP3.y,newP4.y)),min(min(newP5.y,newP
6.y),min(newP7.y,newP8.y)));
 newBottom.z =
min(min(min(newP1.z,newP2.z),min(newP3.z,newP4.z)),min(min(newP5.z,newP
6.z),min(newP7.z,newP8.z)));
 newUp.x =
max(max(max(newP1.x,newP2.x),max(newP3.x,newP4.x)),max(max(newP5.x,newP
6.x),max(newP7.x,newP8.x)));
 newUp.y =
max(max(max(newP1.y,newP2.y),max(newP3.y,newP4.y)),max(max(newP5.y,newP
6.y),max(newP7.y,newP8.y)));
 newUp.z =
max(max(max(newP1.z,newP2.z),max(newP3.z,newP4.z)),max(max(newP5.z,newP
6.z),max(newP7.z,newP8.z)));

 int newWidth = newUp.x - newBottom.x;
 int newHeight= newUp.y - newBottom.y;
    int newDepth = newUp.z - newBottom.z;


 /////////// Rotate /////////////////
 double xold1=c11*(newBottom.x-1)+c21*(newBottom.y-1)+c31*(newBottom.z-
1);
 double yold1=c12*(newBottom.x-1)+c22*(newBottom.y-1)+c32*(newBottom.z-
1);
```

```
 double zold1=c13*(newBottom.x-1)+c23*(newBottom.y-1)+c33*(newBottom.z-
1);
 int BoxWidthHeight = m_Boxwidth()*m_Boxheight();
 for (znew=newBottom.z; znew<=newUp.z; znew++)
 {
   xold1 += c31;
   yold1 += c32;
   zold1 += c33;

   double xold2 = xold1;
   double yold2 = yold1;
   double zold2 = zold1;

   for (ynew=newBottom.y; ynew<=newUp.y; ynew++)
   {
     xold2 += c21;
     yold2 += c22;
     zold2 += c23;

     xold = xold2;
     yold = yold2;
     zold = zold2;

     for (xnew=newBottom.x; xnew<=newUp.x; xnew++)
     {
       xold +=c11;
       yold +=c12;
       zold +=c13;

       if ( (zold<bottom) || (zold>top) || (yold<bottomRight.y) ||
(yold>topLeft.y) || (xold<topLeft.x) || (xold>bottomRight.x) )
         continue;
       int indexold =  int(zold-bottom)*BoxWidthHeight +int(yold-
bottomRight.y)*m_Boxwidth() +int(xold-topLeft.x);

       if  (m_ArrayVoxel[indexold] == 1)
       {
         x=xnew+centerx(cx);
         y=ynew+centery(cy);
         z=znew+centerz(cz);

         //  if ( (x<0) || (x>=width) || (y<0) || (y>=height) || (z<0)
|| (z>=depth)) return 1000000; //no particles on box boarder.

         if (x<0) x+=width;
         if (x>=width) x-=width;

         if (y<0) y+=height;
         if (y>=height) y-=height;

         if (z<0) z+=depth;
         if (z>=depth) z-=depth;

         int index = z*width*height + y*width +x;


         if (im[index]>0)
```

```
          {
            overlap++;

            if (overlap > overlapped)
              return overlap;
            if(fill)
              im[index]+=ID;
          }
          else
          {
            if(fill)
            {
              im[index]=ID;
              filled++;
            }
          }
        }
      }//end for xnew
    }//end for ynew
  }//end for znew

  if (fill)
    return filled;
  else
    return overlap;
}

///////// private KLinkedList implementations ////////////////////

K3DConnectedComponentLabeler::KLinkedList::KLinkedList()
{
  this->header = NULL;
  this->regionCount = 0;
}

K3DConnectedComponentLabeler::KLinkedList::~KLinkedList()
{
  KNode* ptr1 = header;
  KNode* ptr2 = header;
  KNode* ptr3 = header;

  if( header != NULL ) {
    do
    {
      do
      {
        if (ptr2->sgNext != NULL){
          ptr3 = ptr2;
          ptr2 = ptr2->sgNext;
        } else if( ptr1->sgNext != NULL ) {
          delete ptr2;
          if( ptr3 != NULL )
            ptr3->sgNext=NULL;
          ptr2 = ptr1;
          ptr3 = ptr1;
        }
      }
```

```
      while(ptr1->sgNext !=NULL);

      ptr1=ptr1->ngNext;
      delete ptr2;
      ptr2=ptr1;
      ptr3=ptr1;
    }
    while(ptr1!=NULL);
  }
}

void K3DConnectedComponentLabeler::KLinkedList::InsertData(int data)
{
 KNode * ptrTemp = new KNode;
 ptrTemp->data=data;
 ptrTemp->ngNext=header;
 header=ptrTemp;
 regionCount++;
}

void  K3DConnectedComponentLabeler::KLinkedList::InsertData(int  addGroup,
int searchGroup)
{
 if ( addGroup != searchGroup ) {

   KNode* tmp1 = header;
   KNode* ptrAdd ;
   KNode* ptrSearch ;

   Search(addGroup,ptrAdd);
   Search(searchGroup,ptrSearch);

   if ( (ptrSearch != NULL) && (ptrAdd != NULL) && (ptrSearch!=ptrAdd) )
{

     if ( ptrSearch != header ) {

       while( tmp1->ngNext != ptrSearch )
         tmp1=tmp1->ngNext;

       tmp1->ngNext=ptrSearch->ngNext;
     }
     else{
       header=ptrSearch->ngNext;
     }

     while( ptrAdd->sgNext != NULL )
       ptrAdd=ptrAdd->sgNext;

     ptrAdd->sgNext=ptrSearch;
   }
 }

}

void K3DConnectedComponentLabeler::KLinkedList::printTable()
{
```

```
 KNode* ptr1 = header;
 KNode* ptr2 = header;

 FILE* fout;
 fout = fopen("equivalence_table.txt","w+");

 do
 {
   do
   {
     fprintf(fout,"%d\t",ptr2->data);
     ptr2=ptr2->sgNext;
   }
   while(ptr2!=NULL);

   ptr1=ptr1->ngNext;
   ptr2=ptr1;
   fprintf(fout,"\n");
 }
 while(ptr1!=NULL);

 fclose(fout);
}

void K3DConnectedComponentLabeler:: KLinkedList::Search(int data,
K3DConnectedComponentLabeler::KNode* &p )
{
 KNode* ptr1 = header;
 KNode* ptr2 = header;

 do
 {
   do
   {
     if (ptr2->data==data){
       p=ptr1;
       return;
     }
     ptr2=ptr2->sgNext;
   }
   while(ptr2!=NULL);

   ptr1=ptr1->ngNext;
   ptr2=ptr1;
 }
 while(ptr1!=NULL);

 p=ptr1;
}

int K3DConnectedComponentLabeler::SetMask(BYTE *mask, int width, int
height, int depth)
{
 if( m_MaskArray != NULL )
   delete []m_MaskArray;

 m_width     = width;
```

```
 m_height  = height;
 m_depth     = depth;

 int size = width*height*depth;
 try
 {
   m_MaskArray = new DWORD[size];
 }

 catch (...)
 {
   AfxMessageBox("Not enough RAM - SetMask");
   return 1;
 }

 for(int i=0; i<size; i++)
   m_MaskArray[i] = (mask[i]>0)?1:0;
 return 0;
}
void K3DConnectedComponentLabeler::unSetMask()
{
 if( m_MaskArray != NULL )
 {
   delete []m_MaskArray;
   m_MaskArray = NULL;
 }
}
DWORD* K3DConnectedComponentLabeler::GetOutput()
{
 return m_MaskArray;
}

void K3DConnectedComponentLabeler::Clear()
{
 m_Components.clear();
 m_ObjectNumber = 0;
}
```

B.7 3D Cluster Simulation

```
#define WALKSTEP 500
//Ellipsoids smulation
int fnSimuSoid(char *filename, bool simu3d)
{
 return D3(filename,simu3d);
}

template <class T>
T Periodic(T x, T boundary) {
 if (x<0) return x+boundary;
 if (x>=boundary) return x-boundary;
```

```
 return x;
}

void rand_rotation(double x1, double x2, double x3,
                        double *c11, double *c12, double *c13,
                        double *c21, double *c22, double *c23,
                        double *c31, double *c32, double *c33)
{
    double theta = (x1+0.5) * Pi * 2; /* Rotation about the pole (Z).
*/
    double phi   = x2 * Pi * 2; /* For direction of pole deflection. */
    double z     = x3 * 2.0;      /* For magnitude of pole deflection.
*/

/* Compute a vector V used for distributing points over the sphere  */
/* via the reflection I - V Transpose(V).  This formulation of V    */
/* will guarantee that if x[1] and x[2] are uniformly distributed,  */
/* the reflected points will be uniform on the sphere.  Note that V */
/* has length sqrt(2) to eliminate the 2 in the Householder matrix. */

    double r  = sqrt( z );
    double Vx = sin( phi ) * r;
    double Vy = cos( phi ) * r;
    double Vz = sqrt( 2.0 - z );

/* Compute the row vector S = Transpose(V) * R, where R is a simple */
/* rotation by theta about the z-axis.  No need to compute Sz since */
/* it's just Vz.                                                    */

    double st = sin( theta );
    double ct = cos( theta );
    double Sx = Vx * ct - Vy * st;
    double Sy = Vx * st + Vy * ct;

/* Construct the rotation matrix  ( V Transpose(V) - I ) R, which   */
/* is equivalent to V S - R.                                        */

    *c11 = Vx * Sx - ct;
    *c12 = Vx * Sy - st;
    *c13 = Vx * Vz;

    *c21 = Vy * Sx + st;
    *c22 = Vy * Sy - ct;
    *c23 = Vy * Vz;

    *c31 = Vz * Sx;
    *c32 = Vz * Sy;
    *c33 = 1.0 - z;   /* This equals Vz * Vz - 1.0 */
}

class KVoxel
{
public:
 short x,y,z;
};

class KBox
```

```
{
public:
 KBox();
 KBox(int x,int y,int z,int a,int b, int c);
 KBox(int x,int y,int z,int a,int b, int c, double iphi, double itheta,
double ipsi);
 virtual ~KBox();
 int xc,yc,zc;
 int ac,bc,cc;
 double phi, theta, psi;
 bool   fill3D(BYTE *im, int cx, int cy, int cz, int width, int height,
int depth, bool fill=false, bool clear=false);
 bool   fill3D(BYTE *im, int cx, int cy, int cz, int width, int height,
int depth,  double phi, double theta, double psi,  bool fill=false,
bool clear=false);
 std::vector<KVoxel> m_Voxel;
};

KBox::KBox()
{
  xc=yc=zc=ac=bc=cc=0;
  phi=theta=psi=0.0;
}

KBox::~KBox()
{
 m_Voxel.clear();
}

KBox::KBox(int x,int y,int z,int a,int b, int c)
{
 xc=x;
 yc=y;
 zc=z;
 ac=a;
 bc=b;
 cc=c;
 for (int k=-c; k<=c;k++)
   for (int j=-b;j<=b;j++)
     for(int i=-a;i<=a;i++)
     {
       double shell = i*i/double(a)/a+j*j/double(b)/b+k*k/double(c)/c;
       if (shell<=1 && shell>=0.8)
       {
         KVoxel tmp;
         tmp.x=i;
         tmp.y=j;
         tmp.z=k;
         m_Voxel.push_back(tmp);
       }
     }
 phi=theta=psi=0.0;
}

KBox::KBox(int x,int y,int z,int a,int b, int c, double iphi, double
itheta, double ipsi)
{
```

```
 xc=x;
 yc=y;
 zc=z;
 ac=a;
 bc=b;
 cc=c;
 for (int k=-c; k<=c;k++)
   for (int j=-b;j<=b;j++)
     for(int i=-a;i<=a;i++)
     {
       double shell = i*i/double(a)/a+j*j/double(b)/b+k*k/double(c)/c;
       if (shell<=1 && shell>=0.8)
       {
         KVoxel tmp;
         tmp.x=i;
         tmp.y=j;
         tmp.z=k;
         m_Voxel.push_back(tmp);
       }
     }
 phi=iphi;
 theta=itheta;
 psi=ipsi;
}

bool KBox::fill3D(BYTE *im, int cx, int cy, int cz, int width, int
height, int depth, bool fill, bool clear)
{
 int x,y,z;

 std::vector<KVoxel>::iterator iter;
 for (iter=m_Voxel.begin(); iter!=m_Voxel.end(); iter++)
 {
   x=iter->x+cx;
   y=iter->y+cy;
   z=iter->z+cz;

   if (x<0) x+=width;
   if (x>=width) x-=width;

   if (y<0) y+=height;
   if (y>=height) y-=height;

   if (z<0) z+=depth;
   if (z>=depth) z-=depth;

   int index = z*width*height + y*width +x;

   if (clear)
     im[index]=0;
   else
   {
     if (im[index]>0)
       return true;
     else
     {
       if(fill)
```

```
          im[index]=1;
        }
      }
    }
    return false;
}


bool KBox::fill3D(BYTE *im, int cx, int cy, int cz, int width, int
height, int depth,  double phi, double theta, double psi, bool fill,
bool clear)
{
    int x,y,z;
    double c11,c12,c13,c21,c22,c23,c31,c32,c33;

rand_rotation(phi,psi,theta,&c11,&c12,&c13,&c21,&c22,&c23,&c31,&c32,&c3
3);

    std::vector<KVoxel>::iterator iter;
    for (iter=m_Voxel.begin(); iter!=m_Voxel.end(); iter++)
    {
      x= cx + int(iter->x*c11 + iter->y*c12 +iter->z*c13+0.5);
      y= cy + int(iter->x*c21 + iter->y*c22 +iter->z*c23+0.5);
      z= cz + int(iter->x*c31 + iter->y*c32 +iter->z*c33+0.5);

      if (x<0) x+=width;
      if (x>=width) x-=width;

      if (y<0) y+=height;
      if (y>=height) y-=height;

      if (z<0) z+=depth;
      if (z>=depth) z-=depth;

      int index = z*width*height + y*width +x;

      if (clear)
        im[index]=0;
      else
      {
        if(fill)
          im[index]=1;
        else
        {
          if (im[index]>0)
            return true;
        }
      }
    }
    return false;
}

struct list_elem_3D
{
    double x,y,z,a,b,c,phi,theta,psi;
    list_elem_3D* next;
};
```

```
class list_3D
{
 list_elem_3D* h;
public:
 list_3D() {h=0;};
 ~list_3D() {release();}
 void add(double x, double y, double z, double a, double b,double c,
double phi, double theta, double psi);
 void del() { list_elem_3D* temp=h;
        h=h->next;
        delete temp;}
 list_elem_3D* first() {return (h); }
 bool intersect(double x, double y, double z,double a, double b,double
c, double phi, double theta, double psi);
 void pr_list_3D(FILE *fp, double global_x, double global_y, double
global_z);
 void release();
};

void list_3D::add(double x, double y, double z, double a, double
b,double c, double phi, double theta, double psi)
{
 list_elem_3D* temp = new list_elem_3D;
 temp->next=h;
 temp->x = x;
 temp->y = y;
 temp->z = z;
 temp->a = a;
 temp->b = b;
 temp->c = c;
 temp->phi = phi;
 temp->theta = theta;
 temp->psi = psi;
 h=temp;
}

bool list_3D::intersect(double x, double y, double z, double a, double
b,double c, double phi, double theta, double psi)
{
 list_elem_3D* temp=h;
 bool not_intersect=true;
 double dx, dy, dz, dia, p_x, p_y, p_z, p_xrot, p_yrot, p_zrot;
 double c11,c12,c13,c21,c22,c23,c31,c32,c33;
 double a11,a12,a13,a21,a22,a23,a31,a32,a33;

 rand_rotation(phi,psi,theta,&a11,&a12,&a13,
             &a21,&a22,&a23,
             &a31,&a32,&a33);
 while ( (temp!=0) && not_intersect)
 {
   rand_rotation(temp->phi,temp->psi,temp->theta,&c11,&c12,&c13,
                         &c21,&c22,&c23,
                         &c31,&c32,&c33);
   dx = temp->x-x;
   dy = temp->y-y;
   dz = temp->z-z;
```

```
   dia = temp->a+a;
   not_intersect = ( (dx*dx + dy*dy + dz*dz) > dia*dia );

   if (!not_intersect)
   {
     not_intersect=true;
         int beta = 0;
     while ((beta<=180) && not_intersect)
     {
       int alpha = -180;
         while ((alpha<=180) && not_intersect)
       {
          p_x =  temp->a * cos(beta*Pi/180);
          p_y =  temp->b * cos(alpha*Pi/180)*sin(beta*Pi/180);
          p_z =  temp->c * sin(alpha*Pi/180)*sin(beta*Pi/180);

          p_xrot= dx + p_x*c11 + p_y*c12 +p_z*c13;
          p_yrot= dy + p_x*c21 + p_y*c22 +p_z*c23;
          p_zrot= dz + p_x*c31 + p_y*c32 +p_z*c33;

          p_x= p_xrot*a11 + p_yrot*a21 + p_zrot*a31;
          p_y= p_xrot*a12 + p_yrot*a22 + p_zrot*a32;
          p_z= p_xrot*a13 + p_yrot*a23 + p_zrot*a33;

         double aa = 1-(p_x*p_x)/(a*a);
         if (aa > 0)
         {
             double bc = sqrt((b*b- c*c)*aa);
             double ba = b*sqrt(aa);
             not_intersect =( (sqrt((p_y-bc)*(p_y-bc)+p_z*p_z)
+sqrt(p_z*p_z+(p_y+bc)*(p_y+bc)))  > (2*ba+1));
         }
         alpha+=20;
       }
       beta+=20;
     }
     if (not_intersect)//check if object is totally inside of the temp
     {
       p_xrot = -dx;
       p_yrot = -dy;
       p_zrot = -dz;
       p_x=p_xrot*c11 + p_yrot*c21 + p_zrot*c31;
       p_y=p_xrot*c12 + p_yrot*c22 + p_zrot*c32;
       p_z=p_xrot*c13 + p_yrot*c23 + p_zrot*c33;

       not_intersect =( (p_x/temp->a)*(p_x/temp->a)
         +(p_y/temp->b)*(p_y/temp->b)
         +(p_z/temp->c)*(p_z/temp->c) > 1.0);
     }
   }
   temp=temp->next ;
 }
 return (!not_intersect);
}

void list_3D::pr_list_3D(FILE *fp, double global_x, double global_y,
double global_z)
```

```
{
 list_elem_3D* temp=h;
 while (temp!=0)
 {
   fprintf(fp, "\n%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f",global_x+temp-
>x,global_y+temp->y,global_z+temp->z,temp->a,temp->b,temp->c,temp-
>phi,temp->theta,temp->psi);
   temp=temp->next ;
 }
}
void list_3D::release()
{
 while (h!=0)
   del();
}
class matrix_3D
{
 list_3D*** p;
 long s1, s2, s3;
public:
 long ub1, ub2, ub3;
 matrix_3D(long d1, long d2, long d3);
 ~matrix_3D();
 list_3D& element(long i, long j, long k);
};

matrix_3D::matrix_3D(long d1, long d2, long d3)
{
 s1=d1;
 s2=d2;
 s3=d3;
 p = new list_3D**[s1];
 for (long i =0; i<s1;i++)
 {
   p[i] = new list_3D*[s2];
   for (long j =0; j<s2;j++)
     p[i][j] = new list_3D[s3];
 }

 ub1=s1-1;
 ub2=s2-1;
 ub3=s3-1;
}


matrix_3D::~matrix_3D()
{
 for (long i =0; i<=ub1; i++)
   delete [] p[i];
 delete [] p;
}


list_3D& matrix_3D::element(long i, long j, long k)
{
 long x=i,y=j, z=k;
 if (i<0) x=ub1+1+i;
```

```
 if (i>ub1) x=i-ub1-1;

 if (j<0) y=ub2+1+j;
 if (j>ub2) y=j-ub2-1;

 if (k<0) z=ub3+1+k;
 if (k>ub3) z=k-ub3-1;

 return(p[x][y][z]);
}

//3D ellipsoid simulation
int D3(char *fname, bool simu3d)
{
 long SimID;
 char line[150];
    long Ntot=0;
 double Vv, Vol, PVol=0, Sv, Surfarea=0;
 double Lx, Ly, Lz,cell_size_x,cell_size_y,cell_size_z;
 double sizemax=0;
 char pattenhead[] = "LX LY  LZ  Rotation  ZDistance";
 char pattenhead2[] ="SizeA  SizeB SizeC FREQ";
 struct elipse_size
 {
   double elipa,elipb,elipc;
 };

 elipse_size* size_array;

 int  rotflag;
 long i,j,k,w; //loop variables

 FILE *stream;
 if ((stream = fopen( fname, "rt")) == NULL)
 {
    return 2;
 }
 if (simu3d)
 {
   fgets(line, 150, stream);
   fgets(line, 150, stream);
 }
 fgets( line, 150, stream);
 if (strstr( line, pattenhead ) == NULL)
 {
   fclose(stream);
   return 0;
 }
 double zDis;
 fscanf( stream, "%lf %lf %lf %d %lf",&Lx, &Ly, &Lz, &rotflag, &zDis);

 double siza,sizb,sizc;

 int freq;

 fgets(line, 150, stream);// read newline after fscanf
 fgets( line, 150, stream);
```

```
 if (strstr(line, pattenhead2) == NULL)
 {
   fclose(stream);
   return 0;
 }
 while (!feof(stream))
 {
   fscanf( stream, "%lf %lf %lf %d",&siza, &sizb, &sizc, &freq);
   Ntot += freq;
   if (siza>sizemax)
     sizemax = siza;
 }
 fclose( stream );

 size_array = new elipse_size[Ntot];

 stream = fopen( fname, "rt");
 if (simu3d)
 {
   fgets(line, 150, stream);
   fgets(line, 150, stream);
 }
 fgets( line, 150, stream); //dummy line
 fgets( line, 150, stream);
 fgets( line, 150, stream);
 i=0;
    double p=1.6075;
 double min_siza=Lx,min_sizb=Ly,min_sizc=Lz;
 while (!feof(stream))
 {
   fscanf( stream, "%lf %lf %lf %d",&siza, &sizb,&sizc, &freq);
   if (siza<min_siza) min_siza=siza;
   if (sizb<min_sizb) min_sizb=sizb;
   if (sizc<min_sizc) min_sizc=sizc;

   for (j=0;j<freq;j++)
   {
     size_array[i].elipa=siza;
     size_array[i].elipc=sizc;
     size_array[i++].elipb=sizb;
     PVol+=Pi*siza*sizc*sizb/6.0;

Surfarea+=Pi*4*pow((pow(siza*sizb/4.0,p)+pow(siza*sizc/4.0,p)+pow(sizb*
sizc/4.0,p))/3.0,1/p);
   }
 }
 fclose( stream );
 //Ntot=i; // based on the size distribution the Ntot changes slightly

 //randomize the size distribution
 SimID = (unsigned)time( NULL );
 //SimID = 1003;
 seedMT(SimID);
 double temp_sizea, temp_sizeb, temp_sizec;
 long ran_n;
 for ( i =0;i<Ntot;i++)
 {
```

```
   temp_sizea=size_array[i].elipa;
   temp_sizeb=size_array[i].elipb;
   temp_sizec=size_array[i].elipc;
   ran_n =long(Ntot*random53());
   size_array[i]=size_array[ran_n];
   size_array[ran_n].elipa=temp_sizea;
   size_array[ran_n].elipb=temp_sizeb;
   size_array[ran_n].elipc=temp_sizec;
 }
 //"randomize the size distribution"Completed

 Vol = Lx*Ly*Lz;
 Vv = PVol/Vol;
 Sv = Surfarea/Vol;

 if (Vv < 0.20)
 {//RSA
   cell_size_x=sizemax*3;
   cell_size_y=cell_size_x;
   cell_size_z=cell_size_x;

   long lx,ly, lz; // lx, ly, lz are the no. of cells in x, y and z
directions
   lx= long (Lx/cell_size_x+1);
   ly= long (Ly/cell_size_y+1);
   lz= long (Lz/cell_size_z+1);

   matrix_3D map_3D(lx,ly,lz); // map_3d is the simulation space

   cell_size_x= Lx/lx;
   cell_size_y= Ly/ly;
   cell_size_z= Lz/lz;

   // you can seed with any uint32, but the best are odds in 0..(2^32 -
1)
   SimID = (unsigned)time( NULL );
   //SimID = 1003;
   seedMT(SimID);

   double rac,rbc,rcc,xc,yc,zc,phi,theta,psi; //current variables

   for ( w=0; w<Ntot; w++)
   {
     bool not_intersect = false;
     long ic,jc,kc;
     rac=size_array[w].elipa/2; //current radius
     rbc=size_array[w].elipb/2;
     rcc=size_array[w].elipc/2;

     while (not_intersect==false)
     {
       not_intersect = true;
       xc=lx*random53(); // this gives a random real number (double)
between 0 and lx
       yc=ly*random53(); // this gives a random real number (double)
between 0 and ly
```

```
       zc=lz*random53(); // this gives a random real number (double)
between 0 and lz
       if (rotflag)
       {
         phi=random53();
         //theta=Pi*random53();
         theta=random53();
         psi=random53();
       }
       else
       {
         phi=theta=psi=0;
       }
       ic=long (floor(xc));
       jc=long (floor(yc));
       kc=long (floor(zc));

       for ( i=0;(i<=2)&& not_intersect;i++)
         for ( j=0;(j<=2)&& not_intersect;j++)
           for ( k=0;(k<=2)&& not_intersect;k++)
             not_intersect = !map_3D.element(i+ic-1,j+jc-1,k+kc-
1).intersect( ( xc-ic+(1-i) )*cell_size_x,( yc-jc+(1-
j) )*cell_size_y,( zc-kc+(1-k) )*cell_size_z,rac,rbc,rcc,phi,theta,psi);

       // this is sending the coordinates of the current point in real
scale wrt  (i+ic-1,j+jc-1) as origin
       // note that (xc-ic,yc-jc) is the coordinate of the point in
apparent scale with the cell (ic,jc) as origin.
       // Thus, ( xc-ic+(1-i),yc-jc+(1-j) ) is the coordinate of the
point in apprant scale with the
       // cell (i+ic-1,j+jc-1) as the origin.
       // Finally the factor *cell_size_x and cell_size_y are used to
transforn the scale to real scale
     }
     // this adds a local coordinate of the cell ic, jc  in real scale
     map_3D.element(ic,jc,kc).add( (xc-ic)*cell_size_x, (yc-
jc)*cell_size_y, (zc-kc)*cell_size_z, rac, rbc,rcc,phi,theta,psi);
     nSimu2d=100*w/Ntot;
   //  cout << w+1<< endl;
   }

   stream = fopen( "output.txt", "w");
   //fprintf(stream,"SimID\tLX\tLY\tLZ\tNtot\tVv\tVol\tSurfarea\tSv\n");

fprintf(stream,"%lf\t%f\t%f\t%f\t%d\t%f\t%f\t%f\t%f",zDis,lx*cell_size_
x,ly*cell_size_y,lz*cell_size_z,Ntot,Vv,Vol,Surfarea,Sv);
   for (i=0;i<lx;i++)
   {
     for(j=0;j<ly;j++)
     {
       for(k=0;k<lz;k++)
       map_3D.element(i,j,k).pr_list_3D
(stream,i*cell_size_x,j*cell_size_y,k*cell_size_z);
     }
   }
   fclose(stream);
 }
```

```
 else
 {//MA
   std::vector<KBox> elist;
   double Ncell=0;
   double phi,theta,psi;
   phi=theta=psi=0.0;
   min_siza=min_sizb=min_sizc*=2;
   SimID = (unsigned)time( NULL );
   //SimID = 1003;
   seedMT(SimID);
   for (i=0;i<Ntot;i++)
   {
     if (rotflag)
     {
       phi=Periodic((0.05-0.15*random53()+phi),1.0);
       theta=Periodic((0.05-0.15*random53()+theta),1.0);
       psi=Periodic((0.05-0.15*random53()+psi),1.0);
     }

     KBox
tmpE(0,0,0,int(size_array[i].elipa/2),int(size_array[i].elipb/2),int(si
ze_array[i].elipc/2),phi,theta,psi);
     elist.push_back(tmpE);
     Ncell+=
ceil(size_array[i].elipa/min_siza)*ceil(size_array[i].elipb/min_sizb)*c
eil(size_array[i].elipc/min_sizc);
   }

   double acratio=min_siza/min_sizc;
   double bcratio=min_sizb/min_sizc;

   double cellsizc=floor(pow(Vol/Ncell/acratio/bcratio,.3333));
   double cellsiza=floor(cellsizc*acratio);
   double cellsizb=floor(cellsizc*bcratio);
   if (cellsizc<min_sizc || cellsizb<min_sizb || cellsiza<min_siza)
     return 3;

   BYTE* imout = new BYTE[int(Lx*Ly*Lz)];// output mask
   bool placing = true;
   while (placing)
   {
     for(i=0; i<int(Vol); i++)
     imout[i] = 0;

     double xcell=0;
     double ycell=0;
     double zcell=0;

     bool forloop=true;

     int retry = 0;
     for (i=0;i<Ntot&&forloop;i++)
     {
       elist[i].xc=int(xcell+elist[i].ac);
       elist[i].yc=int(ycell+elist[i].bc);
       elist[i].zc=int(zcell+elist[i].cc);
```

```
        if
(!elist[i].fill3D(imout,elist[i].xc,elist[i].yc,elist[i].zc,int(Lx),int
(Ly),int(Lz),elist[i].phi,elist[i].theta,elist[i].psi))
        {

elist[i].fill3D(imout,elist[i].xc,elist[i].yc,elist[i].zc,int(Lx),int(L
y),int(Lz),elist[i].phi,elist[i].theta,elist[i].psi,true);
        }
        else
          i--;

        zcell+=cellsizc;

        if (zcell>=Lz)
        {
          zcell=0;
          ycell+=cellsizb;
          if (ycell>=Ly)
          {
            ycell=0;
            xcell+=cellsiza;
            if (xcell>=Lx)
            {
              cellsiza-=1;
              cellsizb-=1;
              cellsizc-=1;
              if (cellsizc<min_sizc || cellsizb<min_sizb ||
cellsiza<min_siza)
              {
                if (retry==1)
                  Ntot=i;
                else
                {
                  xcell=1;
                  ycell=1;
                  zcell=1;
                  retry++;
                }
                //return 3;
              }
              else
                forloop=false;
            }
          }
        }
      }
      if (forloop)
        placing = false;
    }
    //random walk

    int xc,yc,zc;

    for(int kk=0;kk<WALKSTEP;kk++)
    {
      for (k=0;k<Ntot;k++)
      {
```

```
        int retry = 0;

elist[k].fill3D(imout,elist[k].xc,elist[k].yc,elist[k].zc,int(Lx),int(L
y),int(Lz),elist[k].phi,elist[k].theta,elist[k].psi,true,true);
        bool walknotallowed=true;
        while(walknotallowed)
        {
          if(retry++>50)
          {
            xc=elist[k].xc;
            yc=elist[k].yc;
            zc=elist[k].zc;
            phi=elist[k].phi;
            theta=elist[k].theta;
            psi=elist[k].psi;
            break;
          }
          walknotallowed=false;
          xc=5-int(11*random53())+elist[k].xc;
          yc=5-int(11*random53())+elist[k].yc;
          zc=5-int(11*random53())+elist[k].zc;

          if (rotflag)
          {
            phi=Periodic((0.1-0.3*random53()+elist[k].phi),1.0);
            //theta=Pi*random53();
            theta=Periodic((0.1-0.3*random53()+elist[k].theta),1.0);
            psi=Periodic((0.1-0.3*random53()+elist[k].psi),1.0);
          }
          else
          {
            phi=theta=psi=0.0;
          }
          if (xc<0) xc+=int(Lx);
          else if (xc>=Lx) xc-=int(Lx);
          if (yc<0) yc+=int(Ly);
          else if (yc>=Ly) yc-=int(Ly);
          if (zc<0) zc+=int(Lz);
          else if (zc>=Lz) zc-=int(Lz);


walknotallowed=elist[k].fill3D(imout,xc,yc,zc,int(Lx),int(Ly),int(Lz),
phi, theta,psi);

        }

        elist[k].xc=xc;
        elist[k].yc=yc;
        elist[k].zc=zc;
        elist[k].phi=phi;
        elist[k].theta=theta;
        elist[k].psi=psi;

elist[k].fill3D(imout,elist[k].xc,elist[k].yc,elist[k].zc,int(Lx),int(L
y),int(Lz),elist[k].phi,elist[k].theta,elist[k].psi,true);

     }
```

```
      nSimu2d=100*kk/WALKSTEP;
    }

    stream = fopen( "output.txt", "w");

fprintf(stream,"%lf\t%f\t%f\t%f\t%d\t%f\t%f\t%f\t%f",zDis,Lx,Ly,Lz,Ntot
,Vv,Vol,Surfarea,Sv);

    for (k=0;k<Ntot;k++)
      fprintf(stream,
"\n%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f\t%f",double(elist[k].xc),double(elist
[k].yc),double(elist[k].zc),
double(elist[k].ac),double(elist[k].bc),double(elist[k].cc),
elist[k].phi,elist[k].theta,elist[k].psi);
 fclose(stream);
 delete [] imout;
  elist.clear();
 }

 delete [] size_array;
 return 1;
}
```

B.8 3D Microstructure Simulation

```
void CImageMaxApp::OnSimu3dpar()
{
 char pattenhead[] = "LX LY  LZ  Vv  VvIn  VvOverlap 2POINT  ELLPBD
NeedDilate  RotAng";
 char line[150];
 CString m_sOpenFileDir= GetProfileString("Settings", "OpenFileDir",
"%USERPROFILE%\\My Documents");

    CFileDialog
dlgFile1(TRUE,"txt","3d_simu.txt",OFN_FILEMUSTEXIST,"Data Files
(*.txt;*.dat)|*.txt; *.dat|All Files (*.*)|*.*||");
 CFileDialog
dlgFile(TRUE,"txt","results_voxels_new.txt.gz",OFN_FILEMUSTEXIST,"Data
Files (*.txt;*.dat;*.gz)|*.txt; *.dat; *.gz|All Files (*.*)|*.*||");

 dlgFile.m_ofn.lpstrTitle="open the particle voxel data file";
 dlgFile1.m_ofn.lpstrTitle="open the 3d simulation data file";

 if (dlgFile1.DoModal() == IDOK)
 if (dlgFile.DoModal() == IDOK)
 {
   CImageMaxDoc *NewDoc=(CImageMaxDoc*)((CImageMaxApp*)AfxGetApp())-
>demoTemplate->OpenDocumentFile(NULL);
   NewDoc->m_fFilename = dlgFile.GetPathName();
   NewDoc->m_fFilename2 = dlgFile1.GetPathName();

   FILE *fp;
   if( ( fp=fopen(dlgFile1.GetPathName(), "r" )) == NULL )
   {
```

```
      AfxMessageBox( "Can't open the 3d simu data file!\n");
      return;
    }

    m_sOpenFileDir= NewDoc->m_fFilename;
    int slashpos = m_sOpenFileDir.ReverseFind('\\');
    if(slashpos != -1)
    {
      m_sOpenFileDir = m_sOpenFileDir.Left(slashpos);
    }

    WriteProfileString("Settings", "OpenFileDir", m_sOpenFileDir );
    SetCurrentDirectory(m_sOpenFileDir);

    fgets( line, 150, fp);
    if (strstr( line, pattenhead ) == NULL)
    {
      fclose(fp);
      AfxMessageBox( "Error data file format!\n");
      return;
    }
    int width,height,depth;
    float dummy;
    int dummy2;
    fscanf(fp, "%d  %d  %d  %f  %f  %f %d %d %d %f",&width, &height,
&depth, &dummy, &dummy, &dummy,&dummy2, &dummy2, &dummy2,&dummy);

    fclose(fp);


    NewDoc->m_fp[0]=(void *)(int)depth;

    CxImage *newout = new CxImage(width,height,24);
    NewDoc->image = newout;
    NewDoc->image->SetStdPalette();

    NewDoc->m_MenuCommand=ID_SIMU_3DPAR;
    NewDoc->hThread=(HANDLE)_beginthread(RunSimuThread,0,NewDoc);

    CString s;
    s.Format("results");
    NewDoc->SetTitle(s);
    NewDoc->UpdateAllViews(0,WM_USER_NEWIMAGE);
  }
}

void RunSimuThread(void *lpParam)
{
  POSITION posView;
  CView *pView;

  CImageMaxDoc *pDoc = (CImageMaxDoc *)lpParam;

  if (pDoc==NULL) return;
  if (pDoc->image==NULL) return;

   //prepare for elaboration
```

```
pDoc->image->SetProgress(0);
pDoc->image->SetEscape(0);

nSimu2d=0;
pDoc->Stopwatch(0);

pDoc->hProgress = (HANDLE)_beginthread(RunSimuProgressThread,0,pDoc);

switch (pDoc->m_MenuCommand)
{
	case ID_SIMU_3DPAR: //3D simulations
	{
	  nSimupart=0;
	int depth = (int)pDoc->m_fp[0];
	int i,ellpbd;
	float Vv, Vvin, Vvoverlap, RotAng;
	DWORD width=pDoc->image->GetWidth();
	DWORD height=pDoc->image->GetHeight();
	char* filename =  (char*)(const char*)pDoc->m_fFilename;
	char* filename2 =  (char*)(const char*)pDoc->m_fFilename2;

	FILE *fp,*fp2;

	fp2 = fopen(filename2, "r" );
	int dummy;
	int NeedDilate;
	char line[150];

	fgets( line, 150, fp2);
	int m;//r of 2point;
	fscanf(fp2, "%d %d  %d  %f  %f  %f %d %d %d %f",&dummy, &dummy,
&dummy, &Vv, &Vvin, &Vvoverlap,&m,&ellpbd,&NeedDilate, &RotAng);
	fgets( line, 150, fp2);// read newline after fscanf
	fgets( line, 150, fp2);
	int isRot;
	fscanf(fp2, "%d %d  %d  %d  %d",&dummy, &dummy, &dummy, &isRot,
&dummy);

	if ((Vv>0.5) || (Vv<0))
	{
	    AfxMessageBox( "Vv too high to simulate!\n");
	    fclose(fp2);
	    break;
	}
	fclose(fp2);

	//get particles voxels
	vector<K3DConnectedComponentLabeler::KBox> objects;

	ifstream ifstream_;
	ifstream_.open(filename, ios::in | ios::binary);
	if (!ifstream_.is_open())
	{
	    AfxMessageBox("Cant open voxel data!");
	    break;
	}
```

```
// create unzipper istream
zlib_stream::zip_istream fpzipper( ifstream_);
int objnumber,x,y,z,tmp,averageArea,maxArea=0,maxAreaID;
unsigned long totalArea=0;

fpzipper>>objnumber;

for(i=0; i<objnumber; i++)
{
    K3DConnectedComponentLabeler::KBox newComponent;

    newComponent.ID = i+1;
    newComponent.bottomRight = CPoint(INT_MIN, INT_MAX);
    newComponent.topLeft     = CPoint(INT_MAX, INT_MIN);
    newComponent.top = INT_MIN;
    newComponent.bottom = INT_MAX;
    newComponent.Area = 0;

    while (1)
    {
        fpzipper>>tmp;
        if (tmp==-100000)
            break;

        x = tmp;
        fpzipper>>y>>z;

        if( z < newComponent.bottom )
            newComponent.bottom = z;

        if( z > newComponent.top )
            newComponent.top = z;

        if( x > newComponent.bottomRight.x )
            newComponent.bottomRight.x = x;

        if( x < newComponent.topLeft.x     )
            newComponent.topLeft.x     = x;

        if( y < newComponent.bottomRight.y )
            newComponent.bottomRight.y = y;

        if( y > newComponent.topLeft.y     )
            newComponent.topLeft.y     = y;

        K3DConnectedComponentLabeler::KVoxel newVoxel;
        newVoxel.x = x;
        newVoxel.y = y;
        newVoxel.z = z;
        newComponent.m_Voxel.push_back(newVoxel);
        newComponent.Area++;
    }
    if (newComponent.Area > maxArea)
    {
        maxArea=newComponent.Area;
        maxAreaID=i;
    }
```

```
            totalArea+=newComponent.Area;
            if (isRot)//Put particle in box for rotation
            {
                int BoxWidthHeight =
newComponent.m_Boxwidth()*newComponent.m_Boxheight();

                newComponent.m_ArrayVoxel = new
BYTE[BoxWidthHeight*newComponent.m_Boxdepth()];

                for (int
j=0;j<BoxWidthHeight*newComponent.m_Boxdepth();j++)
                    newComponent.m_ArrayVoxel[j] = 0;

                vector<K3DConnectedComponentLabeler::KVoxel>::iterator
iter;

                for (iter=newComponent.m_Voxel.begin();
iter!=newComponent.m_Voxel.end(); iter++)
                    newComponent.m_ArrayVoxel[(iter->z-
newComponent.bottom)*BoxWidthHeight

+(iter->y-newComponent.bottomRight.y)*newComponent.m_Boxwidth()
                                                    +(iter->x-
newComponent.topLeft.x)] = 1;
            }
            objects.push_back(newComponent);
        }
        objnumber = int( objects.size());
        averageArea=totalArea/objnumber;
        //particle shape done

        //get cluster
        if( ( fp=fopen( "output.txt", "r" )) == NULL )
        {
            AfxMessageBox( "Can't open cluster data file!\n");
            break;
        }

        int ntot;
        float fdummy;
        fscanf(fp, "%lf %f  %f  %f  %d  %f  %f  %f  %f",&fdummy, &fdummy,
&fdummy, &fdummy, &ntot, &fdummy, &fdummy, &fdummy, &fdummy);

        Ellipsoid *soids = new Ellipsoid[ntot+1]; // +1 to fix problem
of no ellipsoid simulated.

        for (i=0; i<ntot; i++)
        {
            fscanf(fp, "%lf %lf %lf %lf %lf %lf %lf %lf %lf",
                        &soids[i].x, &soids[i].y, &soids[i].z,
                                                        &soids[i].a,
&soids[i].b, &soids[i].c,
                        &soids[i].phi, &soids[i].theta, &soids[i].psi);
            soids[i].size=4/3.0*Pi*soids[i].a*soids[i].b*soids[i].c;
            soids[i].tofill=int(soids[i].size*Vvin);
            soids[i].a-=ellpbd;
            soids[i].b-=ellpbd;
```

```
            soids[i].c-=ellpbd;
            //soids[i].c-=3;
        }
        fclose(fp);
        //get cluster done

        //Simulate 3D microstructure
        int imSize = width*height*depth;
        DWORD* imout;
        try {
            imout = new DWORD[imSize];// output mask
        }
        catch (...)
        {
            AfxMessageBox("RAM");
            break;
        }

        for(i=0; i<imSize; i++)
            imout[i] = 0;

        int xc,yc,zc,vFilled=0,vToFill=int(Vv*imSize);

        int nObject,retry;

        vector<K3DConnectedComponentLabeler::KVoxel> pVoxel;
        K3DConnectedComponentLabeler::KVoxel tmpVoxel;

        //fill cluster using MA method
        //no rotation yet (ntot always ==0)
        FILE *fpout = fopen("results_debug.txt","w");
        long SimID;

        fprintf(fpout,"tofill filled  fillnumber\n");
        fclose(fpout);
        fpout = fopen("results_debug.txt","a");

        for (i=0; i<ntot; i++)
        {
            pVoxel.clear();
            //initial position
            int clusterFilled=0;
            int fillnumber;
            double cellsize=3;
            double xcell,ycell,zcell;
            double c11,c12,c13,c21,c22,c23,c31,c32,c33;

rand_rotation(soids[i].phi,soids[i].psi,soids[i].theta,&c11,&c12,&c13,
                                                    &c21,&c22,&c23,
                                                    &c31,&c32,&c33);

            xcell=-soids[i].a;
            ycell=-soids[i].b;
            zcell=-soids[i].c;

            xc=Periodic(int(soids[i].x+xcell*c11 + ycell*c12 +
zcell*c13),int(width));
```

```
            yc=Periodic(int(soids[i].y+xcell*c21 + ycell*c22 +
zcell*c23),int(height));
            zc=Periodic(int(soids[i].z+xcell*c31 + ycell*c32 +
zcell*c33),depth);

            int j=0;

            SimID = (unsigned)time( NULL );
            seedMT(SimID);
            while (1)
            {
                if(soids[i].isinside(int(xcell),int(ycell),int(zcell)))
                {
                    nObject=int(random53()*objnumber);
                    int objectoverlaplimit =
int(objects[nObject].Area*Vvoverlap/2);
                    if
(objects[nObject].fill3DRotate(imout,xc,yc,zc,width,height,depth,soids[
i].phi,soids[i].theta,soids[i].psi,objectoverlaplimit)<=objectoverlapli
mit)
                    {

clusterFilled+=objects[nObject].fill3DRotate(imout,xc,yc,zc,width,heigh
t,depth,soids[i].phi,soids[i].theta,soids[i].psi,objectoverlaplimit,tru
e);
                        tmpVoxel.x=xc;
                        tmpVoxel.y=yc;
                        tmpVoxel.z=zc;
                        tmpVoxel.ID=nObject;
                        pVoxel.push_back(tmpVoxel);
                        if (clusterFilled>=soids[i].tofill)
                        {
                            fillnumber=j;
                            break;
                        }
                        j++;
                    }
                }
                ycell+=cellsize;

                if (ycell>=soids[i].b)
                {
                    ycell=-soids[i].b;
                    xcell+=cellsize;
                    if (xcell>=soids[i].a)
                    {
                        xcell=-soids[i].a;
                        zcell+=cellsize;
                        if (zcell>=soids[i].c)
                        { fillnumber=j;
                            break;
                        }
                    }
                }

                xc=Periodic(int(soids[i].x+xcell*c11 + ycell*c12 +
zcell*c13),int(width));
```

```
                yc=Periodic(int(soids[i].y+xcell*c21 + ycell*c22 +
zcell*c23),int(height));
                zc=Periodic(int(soids[i].z+xcell*c31 + ycell*c32 +
zcell*c33),depth);
            }

            fprintf(fpout,"%d %d %d\n",
soids[i].tofill,clusterFilled,fillnumber);
            fclose(fpout);
            fpout = fopen("results_debug.txt","a");

            //random walk
            SimID = (unsigned)time( NULL );
            //SimID = 1003;
            seedMT(SimID);
            for(int kk=0;kk<0;kk++)
            {
                for (int k=0;k<pVoxel.size();k++)
                {
                    bool walknotallowed=true;
                    retry=0;
                    int
cleaned=objects[pVoxel[k].ID].fill3D(imout,pVoxel[k].x,pVoxel[k].y,pVox
el[k].z,width,height,depth,true,true);
                    int remain=objects[pVoxel[k].ID].Area-cleaned;
                    if (remain>objects[pVoxel[k].ID].Area*Vvoverlap)
                    {
                            xc=pVoxel[k].x;
                            yc=pVoxel[k].y;
                            zc=pVoxel[k].z;
                            walknotallowed=false;
                    }
                    clusterFilled-=cleaned;

                    while(walknotallowed)
                    {
                        xc=1-int(3*random53())+pVoxel[k].x;
                        yc=1-int(3*random53())+pVoxel[k].y;
                        zc=1-int(3*random53())+pVoxel[k].z;

                        if (xc<0) xc+=width;
                        else if (xc>=width) xc-=width;
                        if (yc<0) yc+=height;
                        else if (yc>=height) yc-=height;
                        if (zc<0) zc+=depth;
                        else if (zc>=depth) zc-=depth;

                        xcell = xc-soids[i].x;
                        if (xcell>soids[i].a) xcell-=width;
                        else if (xcell<-soids[i].a) xcell+=width;
                        ycell = yc-soids[i].y;
                        if (ycell>soids[i].b) ycell-=height;
                        else if (ycell<-soids[i].b) ycell+=height;
                        zcell = zc-soids[i].z;
                        if (zcell>soids[i].c) zcell-=depth;
                        else if (zcell<-soids[i].c) zcell+=depth;
```

```
if(!soids[i].isinside(int(xcell),int(ycell),int(zcell)))
                        {
                            retry;
                            continue;
                        }

                        if
(objects[pVoxel[k].ID].fill3D(imout,xc,yc,zc,width,height,depth)<=objec
ts[pVoxel[k].ID].Area*Vvoverlap)
                            walknotallowed=false;
                        retry++;
                        if (retry>100)
                        {
                            xc=pVoxel[k].x;
                            yc=pVoxel[k].y;
                            zc=pVoxel[k].z;
                            break;
                        }
                    }


clusterFilled+=objects[pVoxel[k].ID].fill3D(imout,xc,yc,zc,width,height
,depth,true);

                    pVoxel[k].x=xc;
                    pVoxel[k].y=yc;
                    pVoxel[k].z=zc;
                }
            }
            vFilled+=clusterFilled;
            fprintf(fpout,"%d %f %f\n",
soids[i].tofill,clusterFilled/soids[i].size,vFilled/float(imSize));
        }

        //Placing particles outside of clusters using RSA Method
        SimID = (unsigned)time( NULL );
        bool maxAreaFilled = false;
        seedMT(SimID);
        while (vFilled<vToFill)
        {
            retry = 0;
            while (1)
            {
                xc=int(width*random53());
                yc=int(height*random53());
                zc=int(depth*random53());

                if (!maxAreaFilled)
                {
                    nObject=maxAreaID;
                    maxAreaFilled=true;
                }
                else
                {
                    nObject=int(random53()*objnumber);
                    if (nObject == maxAreaID && objnumber > 100)
```

```
                        continue;
                }
        ////////////////////////for particle rotation
                if (isRot)
                {
                    double phi=RotAng*random53();
                    double theta=RotAng*random53();
                    double psi=random53();
                    int objectoverlaplimit =
int(objects[nObject].Area*Vvoverlap/2);

                    if
(objects[nObject].fill3DRotate(imout,xc,yc,zc,width,height,depth,phi,th
eta,psi,objectoverlaplimit)<=objectoverlaplimit)
                    {

vFilled+=objects[nObject].fill3DRotate(imout,xc,yc,zc,width,height,dept
h,phi,theta,psi,objectoverlaplimit,true);
                        fprintf(fpout,"%d %d\n",
objects[nObject].Area,vFilled);
                        break;
                    }
                    if (retry++>200){break;}
                }
                else
                {
        ////////////////////////////////
                    if
(objects[nObject].fill3D(imout,xc,yc,zc,width,height,depth)<=objects[nO
bject].Area*Vvoverlap/2)
                    {

vFilled+=objects[nObject].fill3D(imout,xc,yc,zc,width,height,depth,true
);
                        fprintf(fpout,"%d %d\n",
objects[nObject].Area,vFilled);
                        break;
                    }
                    if (retry++>200){break;}
                }
            }

            nSimupart=int(double(vFilled)/vToFill*100)-1;

        }
        //delete voxel array for rotation
        if (isRot)
            for(i=0; i<objnumber; i++)
                delete [] objects[i].m_ArrayVoxel;

        fprintf(fpout,"done!");
        fclose(fpout);

        //output images
        for (i=0; i<depth;i=i+10)
        {
            CString zeros, imgNString, imagename;
```

```
            if (i<10)
                zeros = "00";
            if ((i<100) && (i>=10))
                zeros = "0";
            if (i>=100)
                zeros = "";

            imgNString.Format("%d", i);
            imagename ="Result2_" + zeros + imgNString + ".png";

            pDoc->image->SetColorHead(&imout[i*height*width],true);
            if (NeedDilate)
            {
                pDoc->image->Dilate(3);
                pDoc->image->Erode(3);
            }

            pDoc->image->Save(imagename, CXIMAGE_FORMAT_PNG);
        }

        delete [] imout;
        delete [] soids;
        pVoxel.clear();
        objects.clear();
        break;
      }//end ID_SIMU_3DPAR
    }

 nSimu2d=100;
 pDoc->Stopwatch(1);

 pDoc->hThread=0;
 _endthread();
 return ;
}

class Ellipsoid
{
public:
 Ellipsoid();
 virtual ~Ellipsoid();
 double x,y,z,a,b,c, phi, theta, psi;
 double size;
 int tofill;
 bool isinside(int xc, int yc, int zc);//no rotation yet
 bool Ellipsoid::isinside(int xc, int yc, int zc, double phi, double
theta, double psi);
};

Ellipsoid::Ellipsoid()
{
 x=y=z=a=b=c=phi=theta=psi=0;
 tofill=0;
}
Ellipsoid::~Ellipsoid()
{
}
```

```
bool Ellipsoid::isinside(int xc, int yc, int zc)
{
 if ( xc*xc/a/a + yc*yc/b/b + zc*zc/c/c < 1 )
   return true;
 else
   return false;
}
bool Ellipsoid::isinside(int xc, int yc, int zc, double phi, double
theta, double psi)
{
 double c11,c12,c13,c21,c22,c23,c31,c32,c33;
 rand_rotation(phi,psi,theta,&c11,&c12,&c13,
               &c21,&c22,&c23,
               &c31,&c32,&c33);
 double xrot = xc*c11 + yc*c21 + zc*c31;
 double yrot = xc*c12 + yc*c22 + zc*c32;
 double zrot = xc*c13 + yc*c23 + zc*c33;
 if ( xrot*xrot/a/a + yrot*yrot/b/b + zrot*zrot/c/c < 1 )
   return true;
 else
   return false;
}

template <class T>
T Periodic(T x, T boundary) {
 if (x<0) return x+boundary;
 if (x>=boundary) return x-boundary;
 return x;
};
```

B.9 Dual-Scale Virtual Cycloids

```
void CImageMaxApp::OnDllCycloid()
{
 dlgCycloid dlg;
 if (dlg.DoModal()==IDOK)
 {
   char sBuffer[BUFFERLEN];
   CString svar;

   if (dlg.m_probeType==3)
     svar = "Data Files (*.gz)|*.gz|All Files (*.*)|*.*||";
   else
     svar = "Data Files (*.bmp;*.tif;*.png)|*.bmp; *.tif; *.png|All
Files (*.*)|*.*||";
   CFileDialog
dlgFile(TRUE,NULL,NULL,OFN_ALLOWMULTISELECT|OFN_EXPLORER|OFN_ENABLESIZI
NG,svar);
   dlgFile.m_ofn.lpstrFile = sBuffer;
   dlgFile.m_ofn.lpstrFile[0] = '\0';
   dlgFile.m_ofn.nMaxFile = BUFFERLEN;
   //get mru, if not exist, use "my documents"
   CString m_sOpenFileDir= GetProfileString("Settings",
"OpenFileDirCycloid", "%USERPROFILE%\\My Documents");
   dlgFile.m_ofn.lpstrInitialDir=(LPCSTR) m_sOpenFileDir;
```

```
   if (dlgFile.DoModal() == IDOK)
   {
     int filenum=0;
     CImageMaxDoc *NewDoc=(CImageMaxDoc*)((CImageMaxApp*)AfxGetApp())-
>demoTemplate->OpenDocumentFile(NULL);

     if (dlg.m_probeType==3)
     {
       CxImage *newout = new CxImage(150,50,24);
       NewDoc->image = newout;
       NewDoc->image->SetStdPalette();
     }

     POSITION pos = dlgFile.GetStartPosition();

     while (pos)
     {
       NewDoc->m_Filename_array[filenum] =
dlgFile.GetNextPathName( pos );
       filenum++;
     }
     //save mru
     m_sOpenFileDir= NewDoc->m_Filename_array[0];

     int slashpos = m_sOpenFileDir.ReverseFind('\\');
     if(slashpos != -1)
     {
       m_sOpenFileDir = m_sOpenFileDir.Left(slashpos);
     }
     WriteProfileString("Settings", "OpenFileDirCycloid",
m_sOpenFileDir );

     SetCurrentDirectory(m_sOpenFileDir);

     //order filename
     BubbleSort(NewDoc->m_Filename_array, filenum);

     if (dlg.m_probeType!=3)
       if (!NewDoc->OnOpenDocument(NewDoc->m_Filename_array[0]))
         return;
         NewDoc->m_fp[4]=(void *)(int)dlg.m_OutputImg;
     NewDoc->m_fp[3]=(void *)(int)dlg.m_nCycloids;
     NewDoc->m_fp[2]=(void *)(int)dlg.m_probeType;
     NewDoc->m_fp[1]=(void *)(int)dlg.m_ZdisFile;
     NewDoc->m_fp[0]=(void *)(int)filenum;
     NewDoc->m_MenuCommand=ID_DLL_CYCLOID;
     NewDoc->m_fPixelSize=dlg.m_cyc_pixelsize;
     NewDoc->m_fZDis=dlg.m_zDis;
     NewDoc->m_fFilename = dlg.m_ZdisFilename;
     NewDoc->m_rCycloid =
dlg.m_rCycloidfactor*dlg.m_zDis/dlg.m_cyc_pixelsize;

     NewDoc->hThread=(HANDLE)_beginthread(RunSimuThread,0,NewDoc);

     CString s;
     s.Format("Virtual Cycloids");
```

```
      NewDoc->SetTitle(s);
      NewDoc->UpdateAllViews(0,WM_USER_NEWIMAGE);
    }
  }
}

void RunSimuThread(void *lpParam)
{
 POSITION posView;
 CView *pView;

 CImageMaxDoc *pDoc = (CImageMaxDoc *)lpParam;

 if (pDoc==NULL) return;
 if (pDoc->image==NULL) return;

  //prepare for elaboration
 pDoc->image->SetProgress(0);
 pDoc->image->SetEscape(0);

 nSimu2d=0;
 pDoc->Stopwatch(0);

 pDoc->hProgress = (HANDLE)_beginthread(RunSimuProgressThread,0,pDoc);

 switch (pDoc->m_MenuCommand)
 {
       case ID_DLL_CYCLOID:
       {
           pDoc->MouseMoveUpdate=false; //disable onmousemove
           nSimupart = 0;

           int i,j,l;
           int zDisFile = (int)pDoc->m_fp[1];
           int zDisNum = (int)pDoc->m_fp[0]-1;
           int probeType = (int)pDoc->m_fp[2];
           //number of cycloids per box
           int nCycloid = (int)pDoc->m_fp[3];
           //output cycloids img
           int Outputimg = (int)pDoc->m_fp[4];
           if (Outputimg)
               CreateDirectory("cycloids", NULL);


           double zTotal=0, rCycloid;
           double tmp;
           int interSec=0;
           int interSec2=0;
           double lTotal,lTotal2;
           double sTotal2,sV2;
           double sTotal,sV;
           int colornew;

           if (probeType==3)
           {//Single particle, cant handle various depth file yet
               //get particle voxel
```

```
                if (fabs(pDoc->m_rCycloid/(pDoc->m_fZDis/pDoc-
>m_fPixelSize)-int(pDoc->m_rCycloid/(pDoc->m_fZDis/pDoc-
>m_fPixelSize)+0.5)) > TOLERANCE)
                {
                        AfxMessageBox("rCycloid % zDis !=0");
                        break;
                }
                vector<K3DConnectedComponentLabeler::KBox> objects;
                FILE *fpout;
                fpout = fopen("sv_batch.txt","w");

                fprintf(fpout,"No\tV\tS\tX\tY\tZ\n");

                ifstream ifstream_;
                ifstream_.open(pDoc->m_Filename_array[0], ios::in |
ios::binary);
                if (!ifstream_.is_open())
                {
                    AfxMessageBox("Cant open voxel data!");
                    break;
                }

                // create unzipper istream
                zlib_stream::zip_istream fpzipper( ifstream_);

                int objnumber,x,y,z,tmp;

                fpzipper>>objnumber;

                for(i=0; i<objnumber; i++)
                {
                    K3DConnectedComponentLabeler::KBox newComponent;

                    newComponent.ID = i+1;
                    newComponent.bottomRight = CPoint(INT_MIN, INT_MAX);
                    newComponent.topLeft     = CPoint(INT_MAX, INT_MIN);
                    newComponent.top = INT_MIN;
                    newComponent.bottom = INT_MAX;
                    newComponent.Area = 0;

                    while (1)
                    {
                        fpzipper>>tmp;
                        if (tmp==-100000)
                            break;

                        x = tmp;
                        fpzipper>>y>>z;

                        if( z < newComponent.bottom )
                            newComponent.bottom = z;

                        if( z > newComponent.top )
                            newComponent.top = z;

                        if( x > newComponent.bottomRight.x )
                            newComponent.bottomRight.x = x;
```

```
                        if( x < newComponent.topLeft.x     )
                            newComponent.topLeft.x     = x;

                        if( y < newComponent.bottomRight.y )
                            newComponent.bottomRight.y = y;

                        if( y > newComponent.topLeft.y     )
                            newComponent.topLeft.y     = y;

                        K3DConnectedComponentLabeler::KVoxel newVoxel;
                        newVoxel.x = x;
                        newVoxel.y = y;
                        newVoxel.z = z;
                        newComponent.m_Voxel.push_back(newVoxel);
                        newComponent.Area++;
                    }

                    objects.push_back(newComponent);
                }

                double particle_zDis = pDoc->m_fZDis/pDoc->m_fPixelSize;
                Cycloid *ArrayCycloid = new Cycloid[nCycloid];
                Cycloid *ArrayCycloid2 = new Cycloid[nCycloid];

                double
rotangles[16]={0,0.463647609,0.785398163,1.107148718,

1.570796327,2.034443936,2.35619449,2.677945045,

3.141592654,3.605240263,3.926990817,4.248741371,

4.71238898,5.176036589,5.497787144,5.819537698};
                rCycloid = pDoc->m_rCycloid;
                for(i=0; i<objnumber; i++)
                {
                    if ((objects[i].top-objects[i].bottom) % 2 == 0)
                        zDisNum = objects[i].top-objects[i].bottom+9;
                    else
                        zDisNum = objects[i].top-objects[i].bottom+10;
                    double particle_zTotal = (zDisNum+1)*particle_zDis;
                    int particle_xTotal,particle_yTotal;
                    if ((objects[i].bottomRight.x-
objects[i].topLeft.x) % 2 == 0)
                        particle_xTotal= objects[i].bottomRight.x-
objects[i].topLeft.x+10;
                    else
                        particle_xTotal= objects[i].bottomRight.x-
objects[i].topLeft.x+11;
                    if ((objects[i].topLeft.y-
objects[i].bottomRight.y) % 2 ==0)
                        particle_yTotal = objects[i].topLeft.y-
objects[i].bottomRight.y+10;
                    else
                        particle_yTotal = objects[i].topLeft.y-
objects[i].bottomRight.y+11;
```

```
                    DWORD* imout;
                    imout = new
DWORD[particle_xTotal*particle_yTotal*(zDisNum+1)];
                    for(j=0;
j<particle_xTotal*particle_yTotal*(zDisNum+1); j++)
                        imout[j] = 0;
                    //put particle in box
                    objects[i].fill3D(imout,particle_xTotal/2,
particle_yTotal/2, zDisNum/2, particle_xTotal, particle_yTotal,
(zDisNum+1), true);

                    //random placement of cycloids
                    long SimID = (unsigned)time( NULL );
                    seedMT(SimID);
                    //z is the starting point of cycloid, x,y are center
point of cycloid

                    for (j=0; j<nCycloid; j++)
                    {
                        int tx=int(particle_xTotal*random53());
                        int ty=int(particle_yTotal*random53());
                        int tznum=int((zDisNum+1)*random53());
                        int tangle=int(random53()*16);


ArrayCycloid[j].initial(tx,ty,tznum*particle_zDis,rotangles[tangle],rCy
cloid);

                        int tx2=int(particle_xTotal/2*random53());
                        int ty2=int(particle_yTotal/2*random53());
                        int tznum2=int((zDisNum+2)/2*random53())*2;

ArrayCycloid2[j].initial(tx2,ty2,tznum2*particle_zDis/2,rotangles[int(r
andom53()*16)],rCycloid);
                    }


                    CxImage *imagenew, *imagenextlayer;
                    imagenew = new CxImage(particle_xTotal,
particle_yTotal,24);
                    imagenextlayer = new CxImage(particle_xTotal/2,
particle_yTotal/2,24);
                    imagenextlayer->SetStdPalette();
                    imagenew->SetStdPalette();


                    double layerz=0;
                    int k;
                    double zTotal2=0;
                    int count=0;
                    for  (k=0;layerz<=particle_zTotal+int(5*rCycloid);k++)
                    {
                        posView = pDoc->GetFirstViewPosition();
                        pView = pDoc->GetNextView(posView);

                        if (k>zDisNum)
                        {
```

```
                            count++;
                            k=0;
                        }
                        imagenew-
>SetColorHead(&imout[k*particle_xTotal*particle_yTotal],true);

                        pDoc->image->Copy(*imagenew);
                        if (k%2==0)
                        {
                            pDoc->image->Resample(pDoc->image-
>GetWidth()/2,pDoc->image->GetHeight()/2, 1, imagenextlayer);
                            for (j=0;j<nCycloid;j++)

ArrayCycloid2[j].countIntersect1(imagenextlayer,zTotal2,pDoc->image);
                            zTotal2+=particle_zDis;
                        }
                        //cycloids
                        for (j=0;j<nCycloid;j++)

ArrayCycloid[j].countIntersect1(imagenew,layerz,pDoc->image);

                        layerz+=particle_zDis;

                        SendMessage(pView->m_hWnd,  WM_USER_NEWDRAWS,0,0);

                        //Sleep(111);

                    }
                    interSec=0;
                    interSec2=0;
                    lTotal=4*nCycloid*rCycloid;
                    lTotal2=lTotal;
                    for (j=0; j<nCycloid; j++)
                    {
                        interSec2+=ArrayCycloid2[j].interSec;
                        interSec+=ArrayCycloid[j].interSec;

                    }

                    sTotal =
2*interSec/lTotal*particle_xTotal*particle_yTotal*particle_zTotal*(pDoc
->m_fPixelSize*pDoc->m_fPixelSize);
                    sTotal2 =
2*interSec2/lTotal2*particle_xTotal*particle_yTotal*particle_zTotal*(pD
oc->m_fPixelSize*pDoc->m_fPixelSize)/2;
                    sTotal=sTotal*4/3-sTotal2/3;
                    fprintf(fpout,"%d\t%f\t%f\t%f\t%f\t%f\n",i,
                        objects[i].Area*pDoc->m_fZDis*pDoc-
>m_fPixelSize*pDoc->m_fPixelSize,
                        sTotal,
                        (objects[i].bottomRight.x-
objects[i].topLeft.x+1)*pDoc->m_fPixelSize,
                        (objects[i].topLeft.y-
objects[i].bottomRight.y+1)*pDoc->m_fPixelSize,
                        (objects[i].top-
objects[i].bottom+1)*particle_zDis*pDoc->m_fPixelSize);
```

```
            delete imagenew;
            delete imagenextlayer;

            nSimupart ++;
            delete [] imout;
        }

        delete [] ArrayCycloid;
        delete [] ArrayCycloid2;
        objects.clear();
        fclose(fpout);

        pDoc->image->Clear();
        pDoc->image->DrawString(0,5,20,"Done!",
                        pDoc->image->RGBtoRGBQUAD(RGB(0,255,0)),
                        "New Arial", 16, 300);
        posView = pDoc->GetFirstViewPosition();
        pView = pDoc->GetNextView(posView);
        SendMessage(pView->m_hWnd, WM_USER_NEWDRAWS,0,0);
        break;
    }

    double *zDis = new double[zDisNum+1];
    char* filename =  (char*)(const char*)pDoc->m_fFilename;
    FILE *fp;
    if (zDisFile)
    {
        if( ( fp=fopen( filename, "r" )) == NULL )
        {
            AfxMessageBox( "Can't open the Z distance file!\n");
            break;
        }

        for (i=0;i<zDisNum;i++)
        {
            fscanf(fp, "%lf", &tmp);
            zDis[i] = tmp/pDoc->m_fPixelSize;
                        zTotal+=zDis[i];
        }

        zDis[zDisNum]=0;

        fclose(fp);

    }
    else
    {
        for (i=0;i<=zDisNum;i++)
        {
            zDis[i] = pDoc->m_fZDis/pDoc->m_fPixelSize;
        }

        zTotal = pDoc->m_fZDis/pDoc->m_fPixelSize * (zDisNum+1);

    }
```

```
            DWORD xTotal=pDoc->image->GetWidth();
            DWORD yTotal=pDoc->image->GetHeight();
            double vTotal = zTotal*yTotal*xTotal;
            double ParticleVtotal=0;

            if (probeType==1)
            {//use strait lines
                lTotal=zTotal*yTotal*xTotal+(zDisNum+1)*yTotal*xTotal*2;
                int *colorz = new int[yTotal*xTotal];
                int *colorx = new int[yTotal];
                int *colory = new int[xTotal];

                for (j=0; j<yTotal; j++)
                    for (i=0; i<xTotal; i++)
                        colorz[i+j*xTotal] = pDoc->image-
>GetPixelGray(i,j);


                for (l=0; l<(int)pDoc->m_fp[0];l++)
                {
                    posView = pDoc->GetFirstViewPosition();
                    pView = pDoc->GetNextView(posView);
                    if (!pDoc->LoadImageThread(pDoc-
>m_Filename_array[l]))
                        break;
                    ParticleVtotal+=zDis[l]*pDoc->image->Sum();

                    for (j=0; j<yTotal; j++)
                            colorx[j] = pDoc->image->GetPixelGray(0,j);
                    for (i=0; i<xTotal; i++)
                            colory[i] = pDoc->image->GetPixelGray(i,0);
                    for (j=0; j<yTotal; j++)
                        for (i=0; i<xTotal; i++)
                        {
                            colornew = pDoc->image->GetPixelGray(i,j);
                            if (colornew != colorz[i+j*xTotal])
                            {
                                interSec++;
                                colorz[i+j*xTotal]=colornew;
                            }
                            if (colornew != colory[i])
                            {
                                interSec++;
                                colory[i]=colornew;
                            }
                            if (colornew != colorx[j])
                            {
                                interSec++;
                                colorx[j]=colornew;
                            }
                        }

                SendMessage(pView->m_hWnd, WM_USER_NEWDRAWS,0,0);

                    Sleep(111);
                    nSimupart ++;
                }
```

```
                delete [] colorz;
                delete [] colory;
                delete [] colorx;
            }

            if (probeType==2)
            {//Dual Scale Virtual Cycloids

                if (fabs(pDoc->m_rCycloid/(pDoc->m_fZDis/pDoc-
>m_fPixelSize)-int(pDoc->m_rCycloid/(pDoc->m_fZDis/pDoc-
>m_fPixelSize)+0.5)) > TOLERANCE)
                {
                        AfxMessageBox("rCycloid % zDis !=0");
                        break;
                }
                int imgnumber=0;
                CString imagename;
                CString imgNString;
                rCycloid = pDoc->m_rCycloid;


                //random placement of cycloids
                Cycloid *ArrayCycloid = new Cycloid[nCycloid];
                Cycloid *ArrayCycloid2 = new Cycloid[nCycloid];
                long SimID = (unsigned)time( NULL );
                seedMT(SimID);
                double
rotangles[16]={0,0.463647609,0.785398163,1.107148718,

1.570796327,2.034443936,2.35619449,2.677945045,

3.141592654,3.605240263,3.926990817,4.248741371,

4.71238898,5.176036589,5.497787144,5.819537698};

                //z is the starting point of cycloid, x,y are center
point of cycloid
                double *tz = new double[zDisNum+1];

                tz[0]=0;

                for (i=1;i<=zDisNum;i++)
                {
                    tz[i]=zDis[i]+tz[i-1];
                }
                for (i=0; i<nCycloid; i++)
                {
                    int tx=int(xTotal*random53());
                    int ty=int(yTotal*random53());
                    int tznum=int((zDisNum+1)*random53());
                    int tangle=int(random53()*16);


ArrayCycloid[i].initial(tx,ty,tz[tznum],rotangles[tangle],rCycloid);

                    int tx2=int(xTotal/2*random53());
                    int ty2=int(yTotal/2*random53());
```

```
                    int tznum2=int((zDisNum+2)/2*random53())*2;

ArrayCycloid2[i].initial(tx2,ty2,tz[tznum2]/2,rotangles[int(random53()*
16)],rCycloid);
                }

                CxImage imagenew, imagenextlayer;

                double layerz=0;
                double zTotal2=0;
                int count=0;
                for (i=0;layerz<=zTotal+int(5*rCycloid);i++)
                {
                    posView = pDoc->GetFirstViewPosition();
                    pView = pDoc->GetNextView(posView);

                    if (i>zDisNum)
                    {
                        count++;
                        i=0;
                    }
                    if (!pDoc->LoadImageThread(pDoc-
>m_Filename_array[i]))
                        break;
                    if (count==0)
                        ParticleVtotal+=zDis[i]*pDoc->image->Sum();
                    imagenew.Copy(*(pDoc->image));

                    if (i%2==0)
                    {//resampling

                        pDoc->image->Resample(pDoc->image-
>GetWidth()/2,pDoc->image->GetHeight()/2, 1, &imagenextlayer);
                        for (j=0;j<nCycloid;j++)

ArrayCycloid2[j].countIntersect1(&imagenextlayer,zTotal2,pDoc->image);
                        zTotal2+=zDis[i];
                    }

                    //cycloids
                    for (j=0;j<nCycloid;j++)

ArrayCycloid[j].countIntersect1(&imagenew,layerz,pDoc->image);

                    layerz+=zDis[i];
                    SendMessage(pView->m_hWnd, WM_USER_NEWDRAWS,0,0);

                    //save image//////////////////
                    if (Outputimg)
                    {
                        CreateDirectory("cycloids", NULL);
                        CString zeros;
                        if (imgnumber<10)
                            zeros = "00";
                        if ((imgnumber<100) && (imgnumber>=10))
                            zeros = "0";
                        if (imgnumber>=100)
```

```
                        zeros = "";

                    imgNString.Format("%d", imgnumber++);
                    imagename ="cycloids\\cycloid_" + zeros +
imgNString + ".BMP";
                    pDoc->image->Save(imagename,  CXIMAGE_FORMAT_BMP);
                }
                ////////////////////////////////////
                //Sleep(111);
                nSimupart ++;
            }

            //calc sv
            //sV=sTotal=0;
            lTotal=4*nCycloid*rCycloid;
            lTotal2=lTotal;
            for (i=0; i<nCycloid; i++)
            {
                interSec+=ArrayCycloid[i].interSec;
                interSec2+=ArrayCycloid2[i].interSec;
            }
            delete [] ArrayCycloid;
            delete [] ArrayCycloid2;
            delete [] tz;
            double vTotal2=int(zTotal/2)*pDoc->image-
>GetWidth()*pDoc->image->GetHeight()/4;
            sTotal2 = 2*interSec2/lTotal2*vTotal2*(pDoc-
>m_fPixelSize*pDoc->m_fPixelSize)*4;
            sV2 = 2*interSec2/lTotal2/pDoc->m_fPixelSize/2;

        }

        sTotal = 2*interSec/lTotal*vTotal*(pDoc->m_fPixelSize*pDoc-
>m_fPixelSize);
        sV = 2*interSec/lTotal/pDoc->m_fPixelSize;
        double sT1=0;
        if (probeType==2)
        { sT1=sTotal;
            sTotal=sTotal*4/3-sTotal2/3;
            sV=sTotal/vTotal/pDoc->m_fPixelSize/pDoc-
>m_fPixelSize/pDoc->m_fPixelSize;
        }

        CString strMessage;
        strMessage.Format("The surface area per unit volume is %.5g
per micron\nThe total surface area is %.5g micron\1362\nS1 is %.5g
micron\1362\nTotal particle volume is %.6g micron\1363",
            sV,sTotal,sT1,ParticleVtotal*pow(pDoc->m_fPixelSize,3));

        AfxMessageBox(strMessage,MB_ICONINFORMATION|MB_OK);

        delete [] zDis;
        pDoc->MouseMoveUpdate=true; //enable onmousemove
        break;
    }//end CYCLOID
}
```

```
 //pDoc->image->SetProgress(100);
 nSimu2d=100;
 pDoc->Stopwatch(1);

 pDoc->hThread=0;
 _endthread();
 return ;
}

class MPoint
{
public:
 int x,y;
 double z,angle;
 void input(int xin,int yin)
 {
   x=xin;
   y=yin;
 }
 void input(int xin,int yin, double zin)
 {
   x=xin;
   y=yin;
   z=zin;
 }
 void input(int xin,int yin, double zin, double anglein)
 {
   x=xin;
   y=yin;
   z=zin;
   angle=anglein;
 }
 bool inside(int sidex,int sidey)
 {
   if (x<0 || x>=sidex || y<0 || y>=sidey)
     return false;
   else
     return true;
 }
};

class Cycloid
{
private:

public:
 double a;
 MPoint center; //starting point of cycloid
 double centercosangle;
 double centersinangle;
 int outofboxX, outofboxY;

 double color; //current pixel color
 double clength, segment; //current length = 8a*sin(t/4)^2
 int interSec;
 bool started, stopped;
 //int fuzzy;
```

```
 double startlength, stoplength;

 Cycloid();
 MPoint GetCycloidPoint(double zlayer,int x, int y);
 void initial(int x, int y, double z, double angle, double a);
   ~Cycloid();
 void countIntersect(CxImage *image, double layerz, CxImage *imagenew);
 double totallength() {
   if (started)
   {
     if (!stopped)
       stoplength=clength;
     return stoplength-startlength;//+1;
   }
   else
     return 0;
 }
};

MPoint Cycloid::GetCycloidPoint(double zlayer,int x, int y)
{
 MPoint pout;
 //fix for float calc error.
 if (zlayer<center.z) zlayer=center.z;
 if (zlayer>center.z+2*a) zlayer=center.z+2*a;

 double temp = a*acos(1.0-(zlayer-center.z)/a)-sqrt(2*a*(zlayer-
center.z)-(zlayer-center.z)*(zlayer-center.z));//-PI*a/2.0;

 pout.x= center.x+int(centercosangle*temp);
 pout.y= center.y+int(centersinangle*temp);

 if (abs(pout.x/x)>outofboxX)
 {
   if (color!=0)
   {
     color=0;
     interSec++;
   }
   outofboxX++;
 }
 if (abs(pout.y/y)>outofboxY)
 {
   if (color!=0)
   {
     color=0;
     interSec++;
   }
   outofboxY++;
 }
 pout.x%=x;
 pout.y%=y;

 if (pout.x<0)
   pout.x+=x;
 if (pout.y<0)
   pout.y+=y;
```

```
 return pout;
}
void Cycloid::initial(int x, int y, double z, double angle, double
rCycloid)
{
 a=rCycloid;
 clength=0;
 interSec=0;
 started=false;
 stopped=false;
 //fuzzy=0;
 center.x=x;
 center.y=y;
 center.z=z;
 center.angle=angle;
 centercosangle=cos(center.angle);
 centersinangle=sin(center.angle);
 outofboxX=0;
 outofboxY=0;
}
Cycloid::Cycloid()
{
}
Cycloid::~Cycloid()
{
}

void Cycloid::countIntersect(CxImage *image, double layerz, CxImage
*imagenew)
{
 //fix for float calc error.
 if ( (layerz-center.z<-TOLERANCE) || (layerz-center.z-2*a>TOLERANCE) )
   return;

 MPoint mark=GetCycloidPoint(layerz,image->GetWidth(),image-
>GetHeight());

 double colornew;
 imagenew->DrawCross(mark.x,mark.y,RGB(125,125,125),0);

 colornew=image->GetPixelGray(mark.x,mark.y);

 if (started)
 {
   if (colornew!=color)
   {
     interSec++;
     color=colornew;
   }
 }
 else
 {
   color=colornew;
   started=true;
 }
}
```

REFERENCES

[1] D. W. Cooper, "Random-sequential-packing simulations in three dimensions for spheres," *Phys. Rev. A,* vol. 38, pp. 522-524, 1988.

[2] P. Louis, and A. M. Gokhale, "Computer simulation of spatial arrangement and connectivity of particles in three-dimensional microstructure: Application to model electrical conductivity of polymer matrix composite," *Acta Materialia,* vol. 44, no. 4, pp. 1519-1528, 1996/4, 1996.

[3] S. Ghosh, Z. Nowak, and K. Lee, "Quantitative characterization and modeling of composite microstructures by voronoi cells," *Acta Materialia,* vol. 45, no. 6, pp. 2215-2234, 1997.

[4] S. Yang, A. Tewari, and A. M. Gokhale, "Modeling of non-uniform spatial arrangement of fibers in a ceramic matrix composite," *Acta Materialia,* vol. 45, no. 7, pp. 3059-3069, 1997.

[5] M. D. Rintoul, and S. Torquato, "Reconstruction of the structure of dispersions," *Journal of Colloid and Interface Science,* vol. 186, no. 2, pp. 467-476, 1997/2/15, 1997.

[6] D. Cule, and S. Torquato, "Generating random media from limited microstructural information via stochastic optimization," *Journal of Applied Physics,* vol. 86, no. 6, pp. 3428-3437, 1999/09/15/, 1999.

[7] C. Manwart, S. Torquato, and R. Hilfer, "Stochastic reconstruction of sandstones," *Physical Review E. Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics,* vol. 62, no. 1 B, pp. 893-899, 2000.

[8] N. Yang, J. Boselli, and I. Sinclair, "Simulation and quantitative assessment of homogeneous and inhomogeneous particle distributions in particulate metal matrix composites," *Journal of Microscopy,* vol. 201, no. pt.2, pp. 189-200, 2001/02/, 2001.

[9] N. Sheehan, and S. Torquato, "Generating microstructures with specified correlation functions," *Journal of Applied Physics,* vol. 89, no. 1, pp. 53-60, 2001/01/01/, 2001.

[10] Z. Shan, and A. M. Gokhale, "Digital image analysis and microstructure modeling tools for microstrucutre sensitive design of materials," *International journal of plasticity,* vol. 20, pp. 1347-1370, 2004.

[11] D. M. Saylor, J. Fridy, B. S. El-Dasher *et al.*, "Statistically representative three-dimensional microstructures based on orthogonal observation sections," *Metallurgical and Materials Transactions A: Physical Metallurgy and Materials*

Science, vol. 35 A, no. 7, pp. 1969, 2004.

[12] M. Li, S. Ghosh, O. Richmond *et al.*, "Three dimensional characterization and modeling of particle reinforced metal matrix composites part ii: Damage characterization," *Materials Science and Engineering A,* vol. 266, no. 1-2, pp. 221-240, 1999/6/30, 1999.

[13] J. Lepinoux, and Y. Estrin, "Mechanical behaviour of alloys containing heterogeneously distributed particles: Modelling with delaunay triangulation," *Acta Materialia,* vol. 48, no. 17, pp. 4337-4347, 2000/11, 2000.

[14] M. Geni, and M. Kikuchi, "Damage analysis of aluminum matrix composite considering non-uniform distribution of sic particles," *Acta Materialia,* vol. 46, no. 9, pp. 3125-3133, 1998/5/22, 1998.

[15] M. Sahimi, *Heterogeneous materials i: Linear transport and optical properties*, New York: Springer-Verlag, 2003.

[16] S. Torquato, *Random heterogeneous materials: Microstructure and macroscopic properties*, New York: Springer-Verlag, 2002.

[17] A. Tewari, and A. M. Gokhale, "Nearest neighbor distances in uniform-random poly-dispersed microstructures," *Materials Science and Engineering A,* vol. 396, no. 1-2, pp. 22-27, 2005/4/15, 2005.

[18] A. TSCHESCHEL, "Statistical characterization of tem images of silica-filled rubber," *Journal of Microscopy,* vol. 217, pp. 75, 2005.

[19] G. Jefferson, H. Garmestani, R. Tannenbaum *et al.*, "Two-point probability distribution function analysis of co-polymer nano-composites," *International Journal of Plasticity,* vol. 21, no. 1, pp. 185, 2005.

[20] A. Tewari, and A. M. Gokhale, "Nearest-neighbor distances between particles of finite size in three-dimensional uniform random microstructures," *Materials Science and Engineering A,* vol. 385, no. 1-2, pp. 332, 2004.

[21] S. Qin, C. Chen, G. Zhang *et al.*, "The effect of particle shape on ductility of sicp reinforced 6061 al matrix composites," *Materials Science and Engineering A,* vol. 272, no. 2, pp. 363-370, 1999.

[22] R. Car, and M. Parrinello, "Unified approach for molecular dynamics and density-functional theory," *Physical Review Letters,* vol. 55, no. 22, pp. 2471, 1985.

[23] S. P. A. Gill, M. G. Cornforth, and A. C. F. Cocks, "Modelling microstructure evolution in engineering materials," *International Journal of Plasticity,* vol. 17, no. 4, pp. 669, 2001.

[24] R. F. Sekerka, "Morphology: From sharp interface to phase field models," *Journal*

of Crystal Growth, vol. 264, no. 4, pp. 530-540, 2004/3/31, 2004.

[25] B. Widom, "Random sequential addition of hard spheres to a volume," *The Journal of Chemical Physics,* vol. 44, no. 10, pp. 3888-3894, 1966.

[26] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth *et al.*, "Equation of state calculations by fast computing machines," *Journal of Chemical Physics,* vol. 21, pp. 1087-1092, 1953/06/, 1953.

[27] J. Serra, "Boolean random functions," *Journal of Microscopy,* vol. 156, pp. 41-63, 1989.

[28] C. L. Y. Yeong, and S. Torquato, "Reconstructing random media. Ii. Three-dimensional media from two-dimensional cuts," *Physical Review E. Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics,* vol. 58, no. 1, pp. 224, 1998.

[29] R. Adler, *The geometry of random fields*, New York: Wiley, 1981.

[30] S. Ghosh, Z. Nowak, and K. Lee, "Tessellation-based computational methods for the characterization and analysis of heterogeneous microstructures," *Composites Science and Technology,* vol. 57, no. 9-10, pp. 1187-1210, 1997.

[31] J. Summerscales, F. J. Guild, N. R. L. Pearce *et al.*, "Voronoi cells, fractal dimensions and fibre composites," *Journal of Microscopy,* vol. 201, no. pt.2, pp. 153, 2001.

[32] M. P. Anderson, D. J. Srolovitz, G. S. Grest *et al.*, "Computer simulation of grain growth - i. Kinetics," *Acta Metallurgica,* vol. 32, no. 5, pp. 783-791, 1984/5, 1984.

[33] H. Singh, and A. M. Gokhale, "Visualization of three-dimensional microstructures," *Materials Characterization,* vol. 54, no. 1, pp. 21-29, 2005/1, 2005.

[34] C. Beisbart, T. Buchert, and H. Wagner, "Morphometry of spatial patterns," *Physica A: Statistical Mechanics and its Applications,* vol. 293, no. 3-4, pp. 592, 2001.

[35] E. E. Underwood, *Quantitative stereology*, MA: Addison-Wesley Publishing Co. Reading, 1970.

[36] B. D. Ripley, *Spatial statistics*, New York: John Wiley and Sons, 1981.

[37] P. Louis, and A. M. Gokhale, "Application of image analysis for characterization of spatial arrangements of features in microstructure," *Metallurgical and Materials Transactions A (Physical Metallurgy and Materials Science),* vol. 26A, no. 6, pp. 1449-56, 1995/06/, 1995.

[38] L. S. Pontryagin, *Foundations of combinational topology*: Dover Publications 1999.

[39] S. Yang, A. Tewari, and A. M. Gokhale, "An experimental method for quantitative characterization of spatial distribution of fibers in composites," *Developments in Materials Characterization Technologies. Symposium held during the 28th Annual Technical Meeting of International Metallographic Society.* p. 33.

[40] D. C. Sterio, "The unbiased estimation of number and sizes of arbitrary particles using the disector," *Journal of Microscopy,* vol. 134, no. pt.2, pp. 127, 1984.

[41] A. Tewari, and A. M. Gokhale, "Efficient estimation of number density in opaque material microstructures: The large-area disector," *Journal of Microscopy,* vol. 200, no. pt.3, pp. 277, 2000.

[42] H. Agrawal, A. M. Gokhale, S. Graham *et al.*, "Rotations of brittle particles during plastic deformation of ductile alloys," *Materials Science and Engineering A,* vol. 328, no. 1, pp. 310, 2002.

[43] P. J. Diggle, *Statistical analysis of spatial point patterns*, London: Academic Press, 1983.

[44] D. Stoyan, S. Kendall, and J. Mecke, *Stochastic geometry and its applications*, NY: John Wiley and Sons, 1985.

[45] S. Chandrasekhar, "Stochastic problems in physics and astronomy," *Rev. Mod. Phys.*, vol. 15, pp. 1-89, 1943.

[46] R. T. DeHoff, "A geometrically general theory of diffusion controlled coarsening," *Acta Metallurgica et Materialia,* vol. 39, no. 10, pp. 2349, 1991.

[47] P. J. Wray, O. Richmond, and H. L. Morrison, "Use of the dirichlet tessellation for characterizing and modeling nonregular dispersions of second-phase particles," vol. 16, no. 1, pp. 39, 1983.

[48] J. Lepinoux, and Y. Estrin, "Mechanical behaviour of alloys containing heterogeneously distributed particles: Modelling with delaunay triangulation," *Acta Materialia,* vol. 48, no. 17, pp. 4337, 2000.

[49] Boselli, and Pitcher, "Secondary phase distribution analysis via finite body tessellation," *Journal of Microscopy,* vol. 195, pp. 104-112, 1999.

[50] R. Kumar, and R. K. Ray, "Modeling microstructural heterogeneity in materials by using q-mode factor analysis," *Materials Characterization,* vol. 40, no. 1, pp. 7, 1998.

[51] A. Tewari, M. Dighe, and A. M. Gokhale, "Quantitative characterization of spatial arrangement of micropores in cast microstructures," *Materials Characterization,*

vol. 40, no. 2, pp. 119, 1998.

[52] A. Balasundaram, and A. M. Gokhale, "Quantitative characterization of spatial arrangement of shrinkage and gas (air) pores in cast magnesium alloys," *Materials Characterization,* vol. 46, no. 5, pp. 419, 2001.

[53] T. Ikegami, "Contacts and coordination numbers in a compact of polyhedral particles," *Journal of the American Ceramic Society,* vol. 79, no. 1, pp. 148, 1996.

[54] C.-H. Kuo, and P. K. Gupta, "Rigidity and conductivity percolation thresholds in particulate composites," *Acta Metallurgica et Materialia,* vol. 43, no. 1, pp. 397, 1995.

[55] R. M. German, and L. Yixiong, "Grain agglomeration in liquid phase sintering," *Journal of Materials Synthesis and Processing,* vol. 4, no. 1, pp. 23, 1996.

[56] S. Yang, A. M. Gokhale, and Z. Shan, "Utility of microstructure modeling for simulation of micro-mechanical response of composites containing non-uniformly distributed fibers," *Acta Materialia,* vol. 48, no. 9, pp. 2307, 2000.

[57] Z. Shan, and A. M. Gokhale, "Micromechanics of complex three-dimensional microstructures," *Acta Materialia,* vol. 49, no. 11, pp. 2001, 2001.

[58] M. F. Horstemeyer, and A. M. Gokhale, "Void-crack nucleation model for ductile metals," *International Journal of Solids and Structures,* vol. 36, no. 33, pp. 5029, 1999.

[59] J. W. Leggoe, A. A. Mammoli, M. B. Bush *et al.*, "Finite element modelling of deformation in particulate reinforced metal matrix composites with random local microstructure variation," *Acta Materialia,* vol. 46, no. 17, pp. 6075, 1998.

[60] J. Wulf, T. Steinkopff, and H. F. Fischmeister, "Fe-simulation of crack paths in the real microstructure of an al(6061)/sic composite," *Acta Materialia,* vol. 44, no. 5, pp. 1765, 1996.

[61] A. Borbely, and H. Biermann, "Finite element investigation of the effect of particle distribution on the uniaxial stress-strain behavior of particulate-reinforced metal-matrix composites," *Advanced Engineering Materials,* vol. 2, no. 6, pp. 366, 2000.

[62] L. Sunghak, K. Dongil, and S. Dongwoo, "Microstructure and fracture of sic-particulate-reinforced cast a356 aluminum alloy composites," *Metallurgical and Materials Transactions A (Physical Metallurgy and Materials Science),* vol. 27A, no. 12, pp. 3893, 1996.

[63] B. F. Sorensen, and R. Talreja, "Effects of nonuniformity of fiber distribution on thermally-induced residual stresses and cracking in ceramic matrix composites," *Mechanics of Materials,* vol. 16, no. 4, pp. 351, 1993.

[64] J.-J. Blandin, and R. Dendievel, "Mesoscale model to predict the effect of microstructural heterogeneities on superplastic deformation," *Acta Materialia,* vol. 48, no. 7, pp. 1541, 2000.

[65] K.-H. Hanisch, D. Konig, and D. Stoyan, "The pair correlation function for point and fibre systems and its stereological determination by planar sections," *Journal of Microscopy,* vol. 140, no. pt.3, pp. 361-70, 1985/12/, 1985.

[66] P. B. Corson, "Correlation functions for predicting properties of heterogeneous materials em dash 1. Experimental measurement of spatial correlation functions in multiphase solids," *J Appl Phys,* vol. 45, no. 7, pp. 3159-3164, 1974/7, 1974.

[67] H. L. Frisch, and F. H. Stillinger, "Contribution to the statistical geometric basis of radiation scattering," *Journal of Chemical Physics,* vol. 38, no. 9, pp. 2207, 1963.

[68] A. Tewari, A. M. Gokhale, J. E. Spowart *et al.*, "Quantitative characterization of spatial clustering in three-dimensional microstructures using two-point correlation functions," *Acta Materialia,* vol. 52, no. 2, pp. 307-319, 2004/1/19, 2004.

[69] H. Singh, A. M. Gokhale, S. I. Lieberman *et al.*, "Image based computations of lineal path probability distributions for microstructure representation," *Materials Science and Engineering A,* vol. 474, no. 1-2, pp. 104-111, 2008.

[70] I. Saxl, "Contact distances and random free paths," *Journal of Microscopy-Oxford,* vol. 170, pp. 53-64, Apr, 1993.

[71] D. Stoyan, S. Kendall, and J. Mecke, *Stochastic geometry and its applications, second ed.*, New York: John Wiley and Sons, 1995.

[72] M. V. Kral, M. A. Mangan, G. Spanos *et al.*, "Three-dimensional analysis of microstructures," *Materials Characterization,* vol. 45, no. 1, pp. 17-23, 2000/7, 2000.

[73] B. J. Inkson, M. Mulvihill, and G. Mobus, "3d determination of grain shape in a feal-based nanocomposite by 3d fib tomography," *Scripta Materialia,* vol. 45, no. 7, pp. 753-758, 2001/10/10, 2001.

[74] M. K. Miller, "The development of atom probe field-ion microscopy," *Materials Characterization,* vol. 44, no. 1-2, pp. 11-27, 2000/0, 2000.

[75] A. Borbely, F. F. Csikor, S. Zabler *et al.*, "Three-dimensional characterization of the microstructure of a metal-matrix composite by holotomography," *Materials Science and Engineering A,* vol. 367, no. 1-2, pp. 40-50, 2004/2/25, 2004.

[76] M. V. Kral, and G. Spanos, "Three-dimensional analysis of proeutectoid cementite precipitates," *Acta Materialia,* vol. 47, no. 2, pp. 711, 1999.

[77] A. Tewari, and A. M. Gokhale, “Application of three-dimensional digital image processing for reconstruction of microstructural volume from serial sections,” *Materials Characterization,* vol. 44, no. 3, pp. 259, 2000.

[78] T. T. Elvins, “A survey of algorithms for volume visualization,” *Computer Graphics,* vol. 26, no. 3, pp. 194, 1992.

[79] W. E. Lorensen, and H. E. Cline, “Marching cubes: A high resolution 3d surface construction algorithm,” *Computer Graphics (ACM),* vol. 21, no. 4, pp. 163, 1987.

[80] M. A. Miodownik, “A review of microstructural computer models used to simulate grain growth and recrystallisation in aluminium alloys,” *Journal of Light Metals,* vol. 2, no. 3 SPEC, pp. 125-135, 2002.

[81] M. Kumar, R. Sasikumar, and P. Kesavan Nair, “Competition between nucleation and early growth of ferrite from austenite--studies using cellular automaton simulations,” *Acta Materialia,* vol. 46, no. 17, pp. 6291-6303, 1998/11, 1998.

[82] E. L. F. Hinrichsen, J.; Jossang, T., “Geometry of random sequential adsorption,” *Journal of Statistical Physics,* vol. 44, pp. 793-822, 1986.

[83] J. Feder, “Random sequential adsorption,” *Journal of Theoretical Biology,* vol. 87, no. 2, pp. 237-254, 1980.

[84] D. W. Cooper, “Random-sequential-packing simulations in three dimensions for spheres,” *Physical Review A,* vol. 38, pp. 522-524, 1988.

[85] B. D. Rabideau, and R. T. Bonnecaze, “Computational study of the self-organization of bidisperse nanoparticles,” *Langmuir,* vol. 20, no. 21, pp. 9408, 2004.

[86] M. T. Todinov, “On some limitations of the johnson-mehl-avrami-kolmogorov equation,” *Acta Materialia,* vol. 48, no. 17, pp. 4217, 2000.

[87] J. Quintanilla, “Microstructure and properties of random heterogeneous materials: A review of theoretical results,” *Polymer Engineering and Science,* vol. 39, no. 3, pp. 559, 1999.

[88] M. Bertram, and H. Wendrock, “Characterization of planar local arrangement by means of the delaunay neighbourhood,” *Journal of Microscopy,* vol. 181, no. pt.1, pp. 45-53, 1996/01/, 1996.

[89] J. P. Myles, E. C. Flenley, N. R. J. Fieller *et al.*, “Statistical tests for clustering of second phases in composite materials,” *Philosophical Magazine A (Physics of Condensed Matter, Defects and Mechanical Properties),* vol. 72, no. 2, pp. 515-28, 1995/08/, 1995.

[90] T. A. Witten, Jr., and L. M. Sander, “Diffusion-limited aggregation, a kinetic

critical phenomenon," *Physical Review Letters,* vol. 47, no. 19, pp. 1400-1403, 1981.

[91] H. Singh, A. M. Gokhale, Y. Mao *et al.*, "Computer simulations of realistic microstructures of discontinuously reinforced aluminum alloy (dra) composites," *Acta Materialia,* vol. 54, no. 8, pp. 2131-43, 2006.

[92] T. Ohira, and T. Kishi, "Effect of iron content on fracture toughness and cracking processes in high strength al-zn-mg-cu alloy," *Materials Science and Engineering,* vol. 78, no. 1, pp. 9, 1986.

[93] A. M. Gokhale, N. U. Deshpande, D. K. Denzer *et al.*, "Relationship between fracture toughness, fracture path, and microstructure of 7050 aluminum alloy: Part ii. Multiple micromechanisms-based fracture toughness model," *Metallurgical and Materials Transactions A: Physical Metallurgy and Materials Science,* vol. 29A, no. 4, pp. 1203, 1998.

[94] S. M. El-Soudani, and R. M. Pelloux, "Influence of inclusion content on fatigue crack propagation in aluminum alloys," vol. 4, no. 2, pp. 531, 1973.

[95] H. Agarwal, A. M. Gokhale, S. Graham *et al.*, "Anisotropy of intermetallic particle cracking damage evolution in an al-mg-si base wrought aluminum alloy under uniaxial compression," *Metallurgical and Materials Transactions A: Physical Metallurgy and Materials Science,* vol. 33, no. 11, pp. 3443, 2002.

[96] P. Campestrini, E. P. M. Van Westing, H. W. Van Rooijen *et al.*, "Relation between microstructural aspects of aa2024 and its corrosion behaviour investigated using afm scanning potential technique," *Corrosion Science,* vol. 42, no. 11, pp. 1853, 2000.

[97] A. K. Mukhopadhyay, and A. K. Sharma, "Influence of fe-bearing particles and nature of electrolyte on the hard anodizing behaviour of aa 7075 extrusion products," *Surface & Coatings Technology,* vol. 92, no. 3, pp. 212, 1997.

[98] D. Juul Jensen, N. Hansen, and F. J. Humphreys, "Texture development during recrystallization of aluminium containing large particles," *Acta Metallurgica,* vol. 33, no. 12, pp. 2155, 1985.

[99] J. Harris, "Particle cracking damage evolution in 7075 wrought aluminum alloy under montonic and cyclic loading conditions," Master Thesis, Materials Science and Engineering, Georgia Institute of Technology, 2005.

[100] M. Ren, J. Yang, and H. Sun, "Tracing boundary contours in a binary image," *Image and Vision Computing,* vol. 20, no. 2, pp. 125-131, 2002/2/1, 2002.

[101] A. L. Geiger, and J. A. Walker, "The processing and properties of discontinuously reinforced aluminum composites," *Jom-Journal of the Minerals Metals & Materials Society,* vol. 43, no. 8, pp. 8-15, Aug, 1991.

[102] A. Isalak, *Ferrous powder metallurgy*, Cambridge: Cambridge International Science Publishing, 1997.

[103] R. R. Fard, and F. Akhlaghi, "Effect of extrusion temperature on the microstructure and porosity of a356-sicp composites," *Journal of Materials Processing Technology,* vol. 187, pp. 433-436, Jun 12, 2007.

[104] V. V. B. Prasad, B. V. R. Bhat, Y. R. Mahajan *et al.*, "Effect of extrusion parameters on structure and properties of 2124 aluminum alloy matrix composites," *Materials and Manufacturing Processes,* vol. 16, no. 6, pp. 841-853, 2001.

[105] A. Slipenyuk, V. Kuprin, Y. Milman *et al.*, "The effect of matrix to reinforcement particle size ratio (psr) on the microstructure and mechanical properties of a p/m processed alcumn/sicp mmc," *Materials Science and Engineering a-Structural Materials Properties Microstructure and Processing,* vol. 381, no. 1-2, pp. 165-170, Sep 15, 2004.

[106] J. E. Spowart, B. Maruyama, and D. B. Miracle, "Multi-scale characterization of spatially heterogeneous systems: Implications for discontinuously reinforced metal-matrix composite microstructures," *Materials Science and Engineering A,* vol. 307, no. 1-2, pp. 51-66, 2001.

[107] A. Tewari, A. M. Gokhale, and R. M. German, "Effect of gravity on three-dimensional coordination number distribution in liquid phase sintered microstructures," *Acta Materialia,* vol. 47, no. 13, pp. 3721-34, 1999.

[108] H. Singh, "Computer simulations of realistic microstructures: Implications for simulation based materials design," Ph.D. Dissertation, School Materials Science & Engineering, Georgia Institute of Technology, Atlanta, 2008.

[109] K. Suzuki, I. Horiba, and N. Sugie, "Linear-time connected-component labeling based on sequential local operations," *Computer Vision and Image Understanding,* vol. 89, no. 1, pp. 1-23, 2003.

[110] A. Sreeranganathan, "Realistic micromechanical modeling and simulation of two-phase heterogeneous materials," Ph.D. Dissertation, School of Materials Science and Engineering, Georgia Institute of Technology, Atlanta, 2008.

[111] "Simpleware - converting 3d images into numerical models (http://www.Simpleware.Com)."

[112] C. E. Catlett, "Teragrid: Analysis of organization, system architecture, and middleware enabling new types of applications," in HPC and Grids in Action, Amsterdam, 2007.

[113] O. M. Ivasishin, R. V. Teliovych, V. G. Ivanchenko *et al.*, "Processing, microstructure, texture, and tensile properties of the ti-6al-4v-1.55b eutectic

alloy," *Metallurgical and Materials Transactions A: Physical Metallurgy and Materials Science,* vol. 39, no. 2, pp. 402-416, 2008.

[114] C. T. Inc. "Titanium-boron (ti-b) phase diagram," 10/04/2009, 2009; http://www.calphad.com/titanium-boron.html.

[115] J. L. Murray, P. K. Liao, and K. E. Spear, *Binary alloy phase diagrams,* H. Baker, ed., p. 285, Materials Park, OH: ASM International, 1992.

[116] S. Tamirisakandala, R. B. Bhat, D. J. McEldowney *et al.,* "Strength modeling of a titanium alloy modified with boron and boron+carbon," *Materials Science and Technology 2003 Meeting.* pp. 185-194.

[117] S. Tamirisakandala, R. B. Bhat, V. A. Ravi *et al.*, "Powder metallurgy ti-6al-4v-xb alloys: Processing, microstructure, and properties," *Jom,* vol. 56, no. 5, pp. 60-63, May, 2004.

[118] S. Tamirisakandala, R. B. Bhat, J. S. Tiley *et al.*, "Grain refinement of cast titanium alloys via trace boron addition," *Scripta Materialia,* vol. 53, no. 12, pp. 1421-1426, Dec, 2005.

[119] S. I. Lieberman, A. M. Gokhale, and S. Tamirisakandala, "Reconstruction of three-dimensional microstructures of tib phase in a powder metallurgy titanium alloy using montage serial sectioning," *Scripta Materialia,* vol. 55, no. 1 SPEC. ISS., pp. 63-68, 2006.

[120] S. I. Lieberman, A. M. Gokhale, and S. Tamirisakandala, "Reconstruction of three-dimensional microstructures of tib whiskers in powder processed ti-6al-4v-1b alloys," *Materials Characterization,* vol. 58, no. 6, pp. 527-533, 2007.

[121] S. I. Lieberman, "Microstructural characterization, visualization, and simulation of ti-b materials," Ph.D. Dissertation, School Materials Science & Engineering, Georgia Institute of Technology, Atlanta, 2007.

[122] J. Arvo, "Fast random rotation matrices," *Graphics gems iii,* D. Kirk, ed., pp. 117-120, San Diego: Academic Press Professional, 1992.

[123] J. Lindblad, "Surface area estimation of digitized 3d objects using weighted local configurations," *Image and Vision Computing,* vol. 23, no. 2, pp. 111-122, 2005.

[124] J. Ziegel, and M. Kiderlen, "Estimation of surface area and surface area measure of three-dimensional sets from digitizations," *Image and Vision Computing,* vol. 28, no. 1, pp. 64-77, 2010.

[125] G. Windreich, N. Kiryati, and G. Lohmann, "Voxel-based surface area estimation: From theory to practice," *Pattern Recognition,* vol. 36, no. 11, pp. 2531-2541, 2003.

[126] J. C. Mullikin, and P. W. Verbeek, “Surface area estimation of digitized planes,” *Bioimaging,* vol. 1, no. 1, pp. 6-16, 1993.

[127] R. Klette, and H. Sun, "Digital planar segment based polyhedrization for surface area estimation," *Visual form 2001,* pp. 356-366, 2001.

[128] F. Sloboda, and B. Za'ko, "On approximation of jordan surfaces in 3d," *Digital and image geometry,* pp. 163-174, 2001.

[129] D. Coeurjolly, F. Flin, O. Teytaud *et al.*, "Multigrid convergence and surface area estimation," *Geometry, morphology, and computational imaging,* pp. 119-124, 2003.

[130] F. Flin, J. B. Brzoska, D. Coeurjolly *et al.*, “Adaptive estimation of normals and surface area for discrete 3-d objects: Application to snow binary data from x-ray tomography,” *Ieee Transactions on Image Processing,* vol. 14, no. 5, pp. 585-596, May, 2005.

[131] Y. Kenmochi, and R. Klette, "Surface area estimation for digitized regular solids." pp. 100-111.

[132] C. S. Smith, and L. Guttman, “Measurement of internal boundaries in three-dimensional structures by random sectioning,” *J. Met.,* vol. 5, no. Trans., pp. 81-7, 1953.

[133] Y. Huang, and R. Klette, "A comparison of property estimators in stereology and digital geometry," *Combinatorial image analysis,* pp. 421-431, 2005.

[134] A. M. Gokhale, A. Tewari, and H. Garmestani, “Constraints on microstructural two-point correlation functions,” *Scripta Materialia,* vol. 53, no. 8, pp. 989-993, 2005.

[135] S. Sunde, “Calculation of conductivity and polarization resistance of composite sofc electrodes from random resistor networks,” *Journal of The Electrochemical Society,* vol. 142, no. 4, pp. L50-L52, 1995.

[136] A. M. Gokhale, R. A. Evans, J. L. Mackes *et al.*, “Design-based estimation of surface area in thick tissue sections of arbitrary orientation using virtual cycloids,” *Journal of Microscopy,* vol. 216, no. 1, pp. 25-31, 2004.

[137] A. J. Baddeley, G. H. J. G, and L. M. Cruz-Orive, “Estimation of surface area from vertical sections,” *Journal of Microscopy,* vol. 142, pp. 259-276, 1986.

[138] H. Singh, A. M. Gokhale, A. Tewari *et al.*, “Three-dimensional visualization and quantitative characterization of primary silicon particles in an al-si base alloy,” *Scripta Materialia,* vol. 61, no. 4, pp. 441-444, 2009.